

WHAT IS BUSINESS LOGIC?

- ➊ Private/sensitive data protection
 - Password
 - Personal information
- ➋ Untrusted data sanitization
 - Injection
- ➌ Specific processing flow of data validation
 - Audit accounts for manipulation

EXAMPLE: INJECTION

Tainted data passed to Runtime.exec may cause issues

```
public class MyClass {  
    public void myFunc(HttpServletRequest request) {  
        ...  
        String param = request.getParameter("taintedParam");  
        String cmd = ... + param;  
        ...  
        Runtime r = Runtime.getRuntime();  
        Process p = r.exec(cmd);  
        ...  
    }  
}
```

A

EXAMPLE: INJECTION

Tainted data passed to Runtime.exec may cause issues

```
public class MyClass {  
    public void myFunc(HttpServletRequest request) {  
        ...  
        String param = request.getParameter("taintedParam"); #1  
        String cmd = ... + param;  
        ...  
        Runtime r = Runtime.getRuntime();  
        Process p = r.exec(cmd);  
        ...  
    }  
}
```

B

EXAMPLE: INJECTION

Tainted data passed to Runtime.exec may cause issues

C

```
public class MyClass {  
    public void myFunc(HttpServletRequest request) {  
        ...  
        String param = request.getParameter("taintedParam"); #1  
        String cmd = ... + param;  
        ...  
        Runtime r = Runtime.getRuntime();  
        Process p = r.exec(cmd); #2  
        ...  
    }  
}
```

WHY IS VERIFYING BUSINESS LOGIC IMPORTANT?

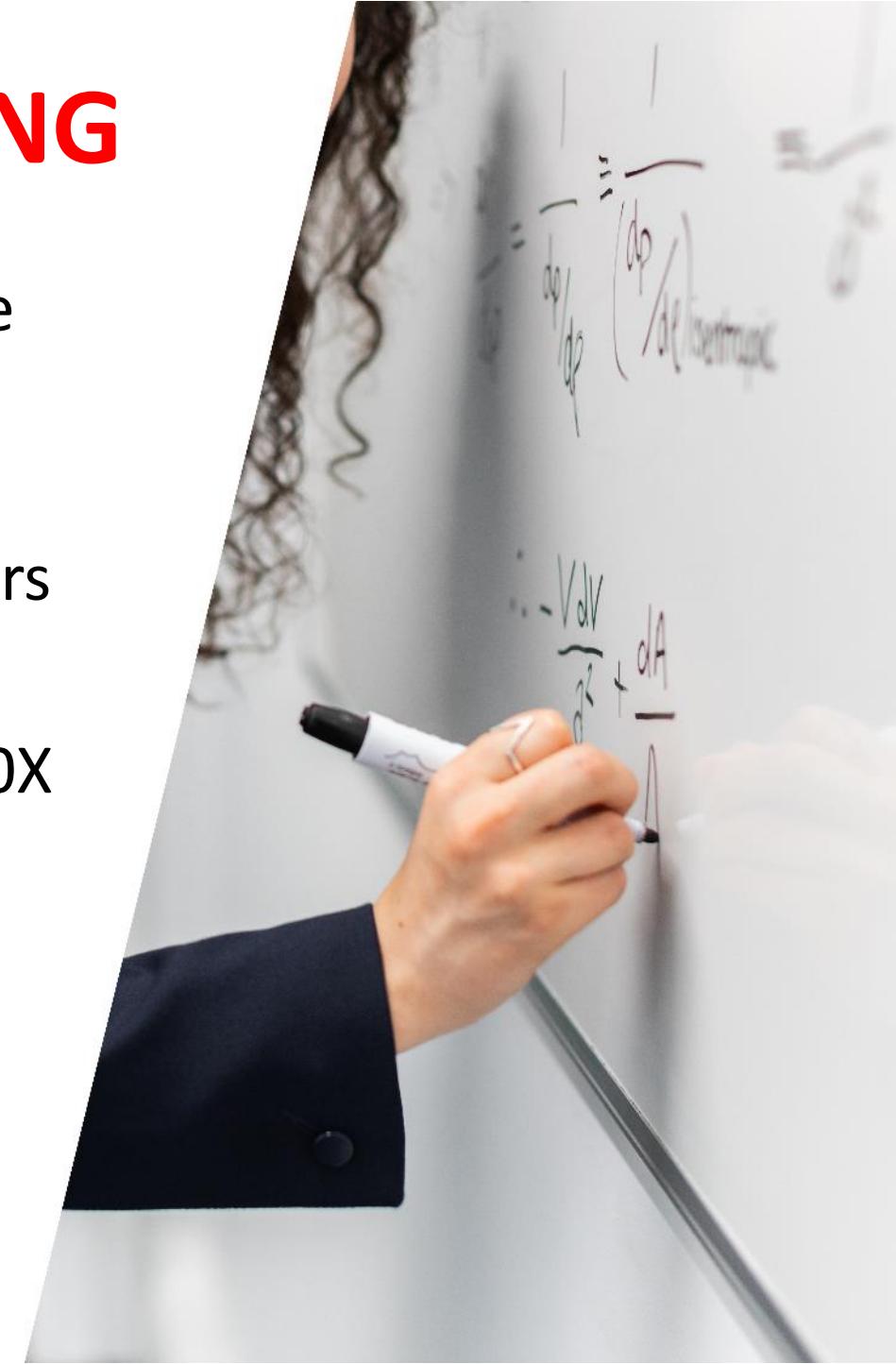
- ✓ In practice, many vulnerabilities result from violations of the underlying business logic
- ✓ Each company has specific business logic so universal checkers/verifiers don't always work

The ability to customize for and verify business logic is needed!

CANDIDATE: THEOREM PROVING

- ✓ Requires strict mathematical methods capable of verification tasks
- ✓ Formalizing business logic into mathematical representation is non-trivial and hard for others to understand
- ✓ ‘Proving’ efforts are very time-consuming (~10X lines of Coq proof code vs. source code)
- ✓ Highly-qualified experts are required

Expensive and unaffordable for most companies!



CANDIDATE: EXISTING SAST TOOLS

- ✓ Focused on revealing common vulnerabilities
 - Null Pointer Dereference (NPD), Use After Free (UAF), Use Uninitialized Variable (UIV)...
- ✓ Compiler based (type or data flow) techniques
 - Effective to discover vulnerabilities related to program syntax and feature
 - But not effective to discover vulnerabilities due to program semantics and business logic
- ✓ Few allow customization and only on pattern match rules only
 - Therefore, limited support





PROGRAM ANALYSIS TECHNIQUES

✓ Analyze based on SSA IRs

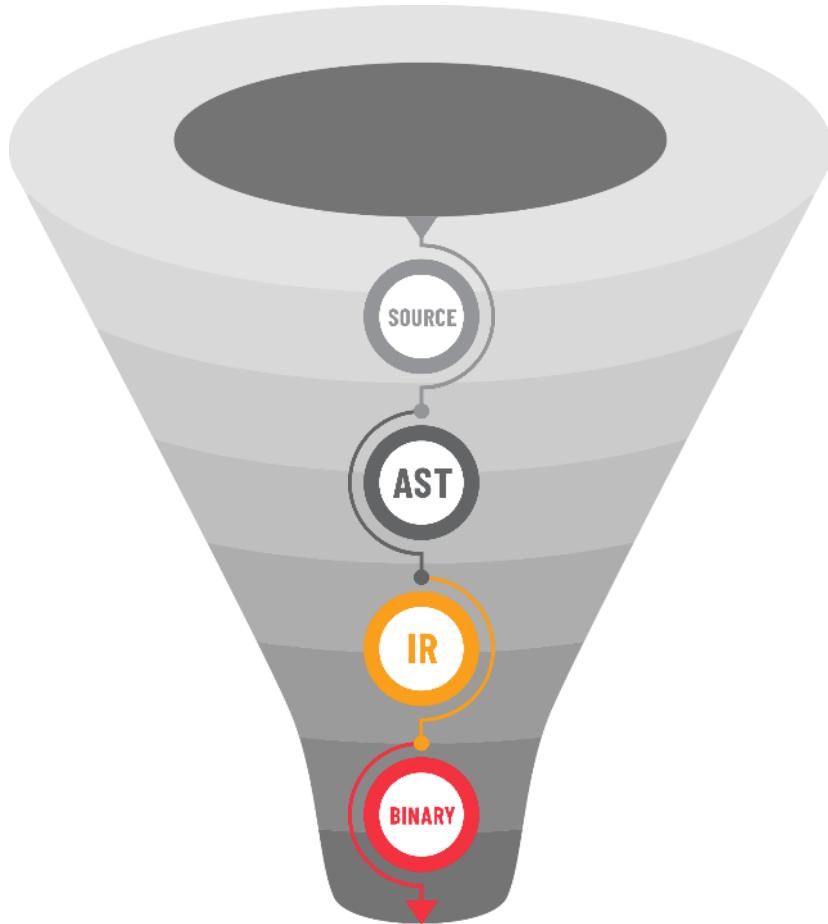
- Context sensitive
- Flow sensitive
- Cross file
- Cross language

✓ On demand analysis

- Less time
- Less memory

✓ Symbolic evaluation

- User customizable rules



xcalscan VERIFYING BUSINESS LOGIC

- ✓ A symbolic evaluation framework
 - APIs to model side effects & to analyze user-defined rules
- ✓ No modification needed in customers' source code
- ✓ Supports analysis without complete source code due to 3rd party API
- ✓ Customers can define their own rules via the same programming language they use during development



EXAMPLE: INJECTION - DEFECT

Tainted data pass to Runtime.exec may cause issues

```
public class MyClass {  
    public void myFunc(HttpServletRequest request) {  
        ...  
        String param = request.getParameter("taintedParam"); #1  
        String cmd = ... + param;  
        ...  
        Runtime r = Runtime.getRuntime();  
        Process p = r.exec(cmd); #2  
        ...  
    }  
}
```

EXAMPLE: INJECTION - REMEDIATION

Sanitize tainted parameter before execution

```
public class MyClass {  
    public void myFunc(HttpServletRequest request) {  
        ...  
        String param = request.getParameter("taintedParam");  
        String sanitizedParam = mySanitizer(param);  
        String cmd = ... + sanitizedParam;  
        ...  
        Runtime r = Runtime.getRuntime();  
        Process p = r.exec(cmd);  
        ...  
    }  
}
```

INJECTION – BUILD THE RULE, STEP 1

```
public class MyClass {  
    public void myFunc(HttpServletRequest request) {  
        ...  
        String param = request.getParameter("taintedParam");  
        String sanitizedParam = mySanitizer(param);  
        String cmd = ... + sanitizedParam;  
        ...  
        Runtime r = Runtime.getRuntime();  
        Process p = r.exec(cmd);  
        ...  
    }  
}
```

#1 Recognize tainted variable

```
public interface ServletRequest {  
    default public String getParameter(String var) {  
        SEE.SideEffect(SEE.SetAttr(See.FuncRet(), "tainted"));  
        ...  
    }  
}
```

INJECTION – BUILD THE RULE, STEP 2

```
public class MyClass {  
    public void myFunc(HttpServletRequest request) {  
        ...  
        String param = request.getParameter("taintedParam");  
        String sanitizedParam = mySanitizer(param);  
        String cmd = ... + sanitizedParam;  
        ...  
        Runtime r = Runtime.getRuntime();  
        Process p = r.exec(cmd);  
        ...  
    }  
}
```

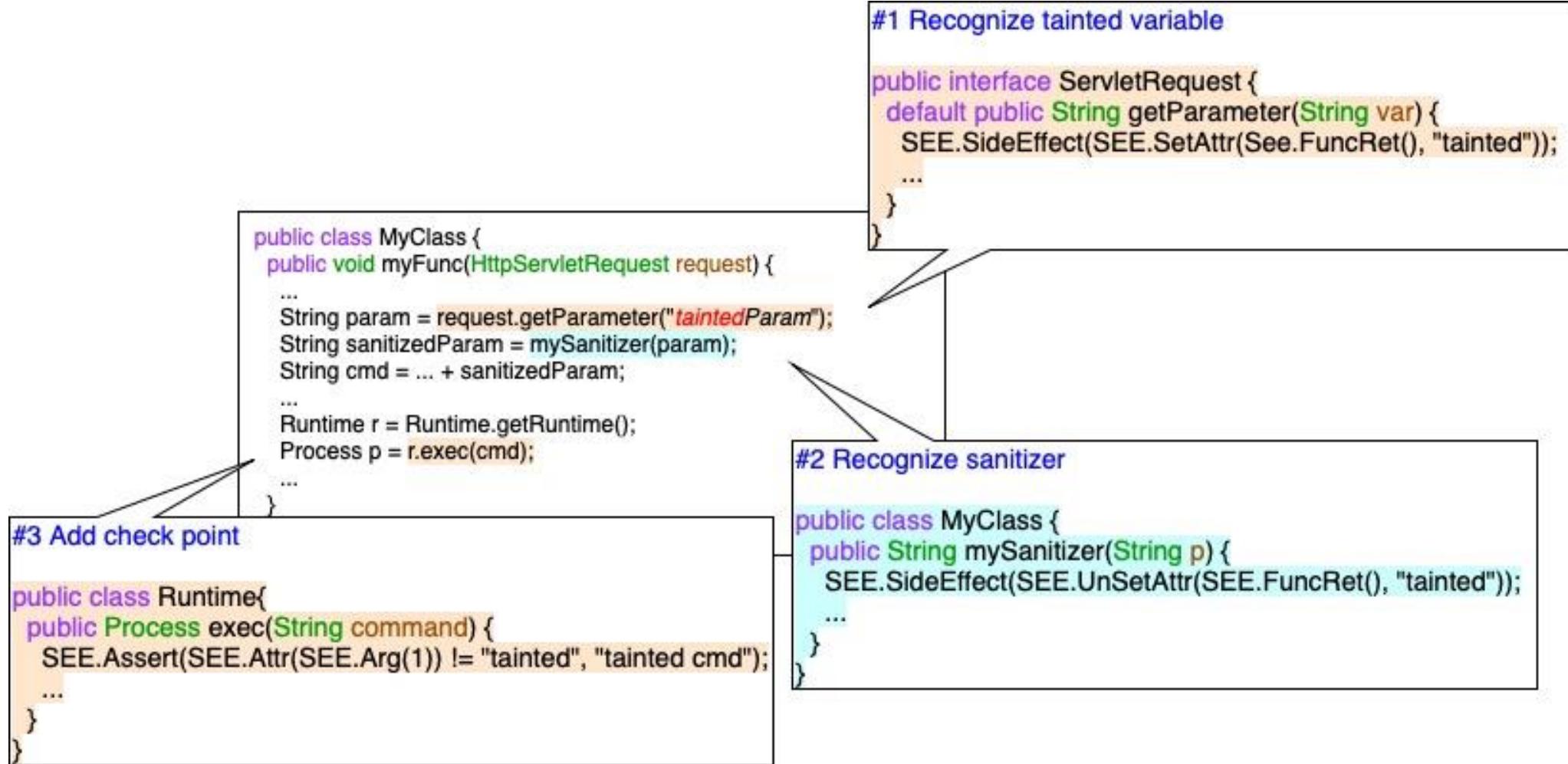
#1 Recognize tainted variable

```
public interface ServletRequest {  
    default public String getParameter(String var) {  
        SEE.SideEffect(SEE.SetAttr(See.FuncRet(), "tainted"));  
        ...  
    }  
}
```

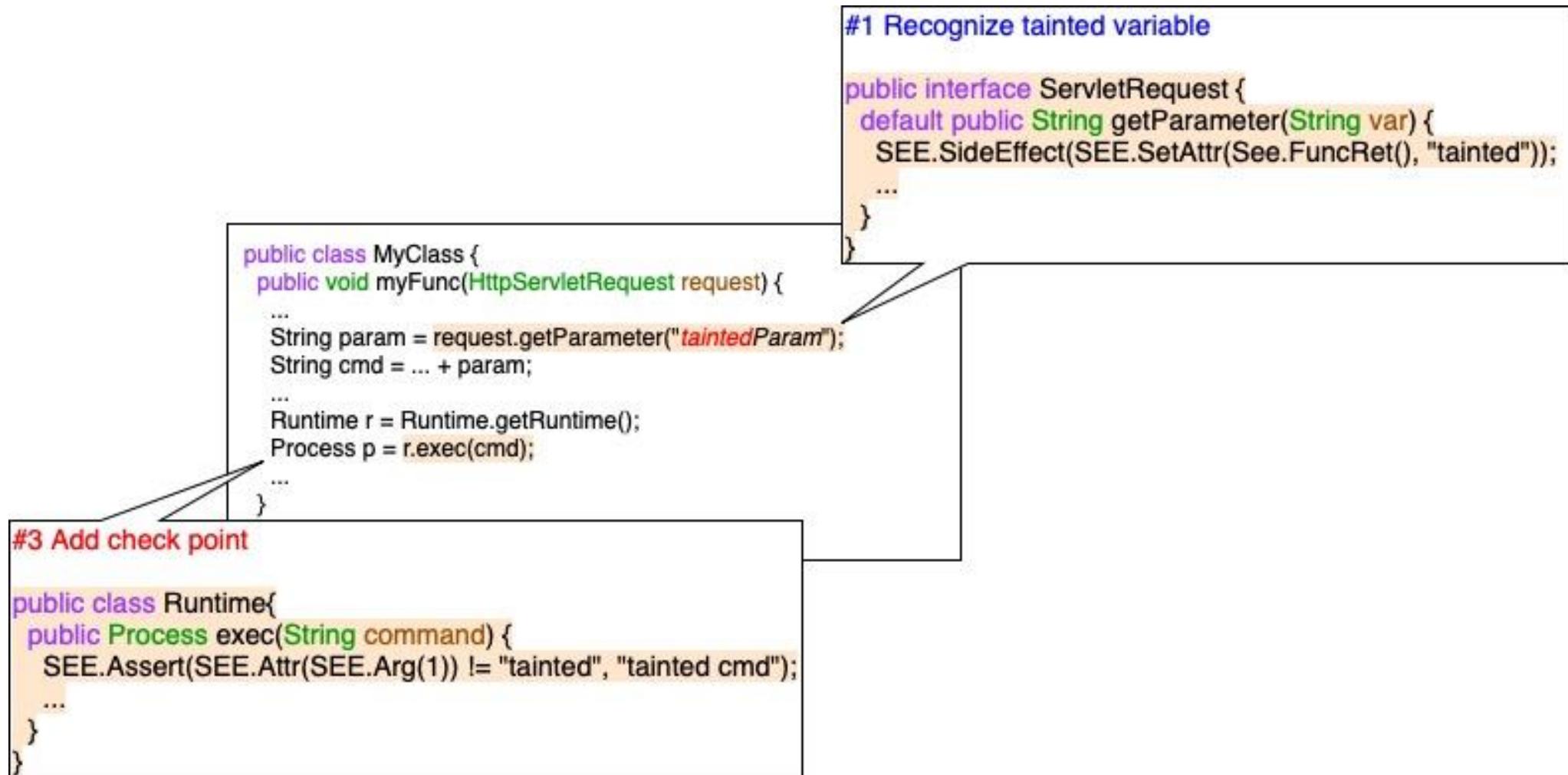
#2 Recognize sanitizer

```
public class MyClass {  
    public String mySanitizer(String p) {  
        SEE.SideEffect(SEE.UnSetAttr(SEE.FuncRet(), "tainted"));  
        ...  
    }  
}
```

INJECTION – BUILD THE RULE, STEP 3

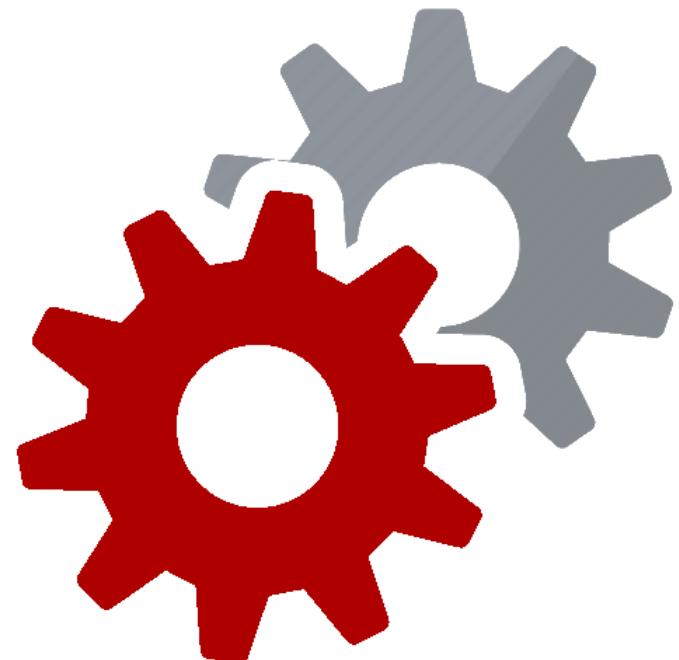


INJECTION – IDENTIFY THE VIOLATION



xcalscan EASY CUSTOMIZATION

- A symbolic evaluation framework
 - Boolean expression: $a > b \dots$
 - Programming concepts: types, super class ...
 - Business logic: call sequence, call depth limitation ...
- We define semantic descriptor for standard runtime libraries
- User functions can be bound to rules available as well
- Selectable compliance standard checks
 - CERT, CWE, OWASP ...
- Customizable locations to perform checking to stay focused
 - xxCoin account overflow/underflow check





COMMON VULNERABILITIES

✓ Analyze based on SSA IRs

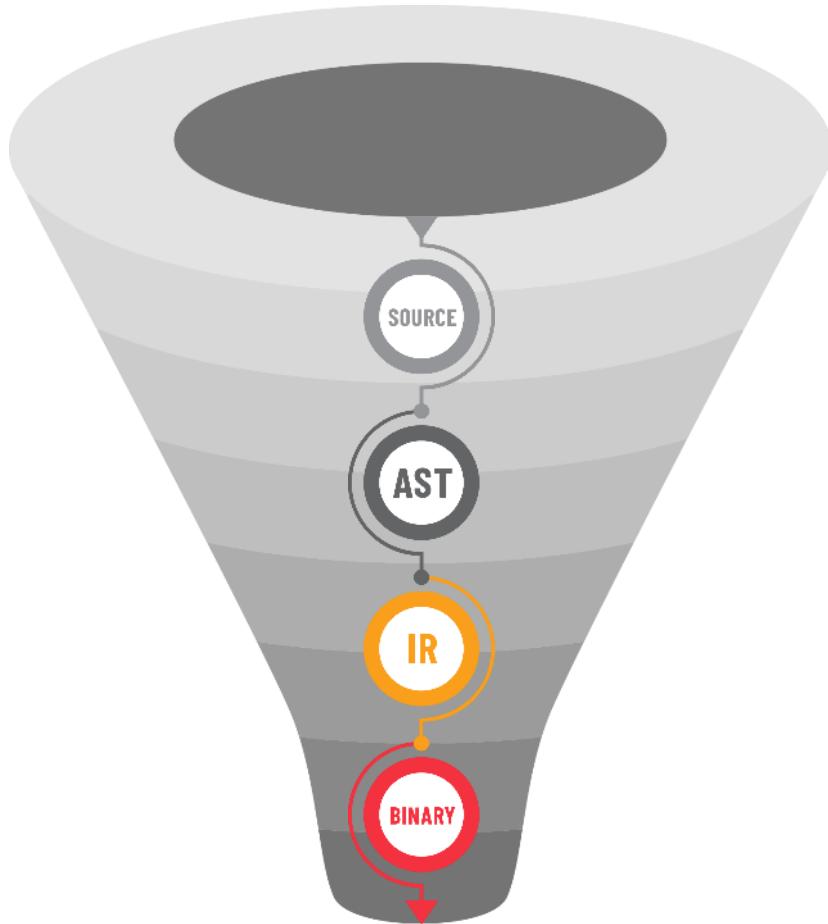
- Context sensitive
- Flow sensitive
- Cross file
- Cross language

✓ On demand analysis

- Less time
- Less memory

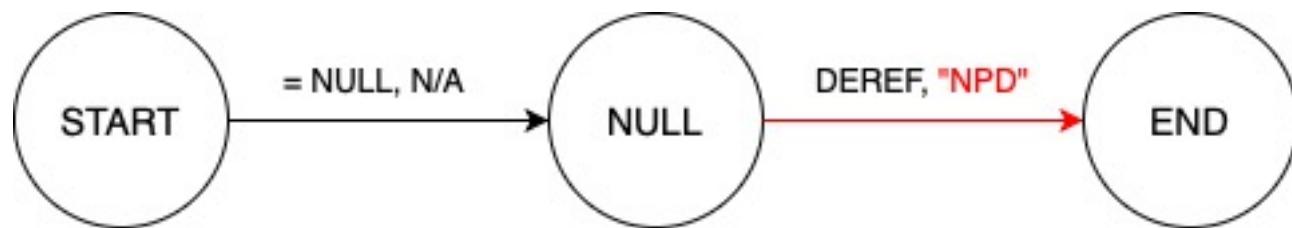
✓ Symbolic evaluation

- User customizable rules



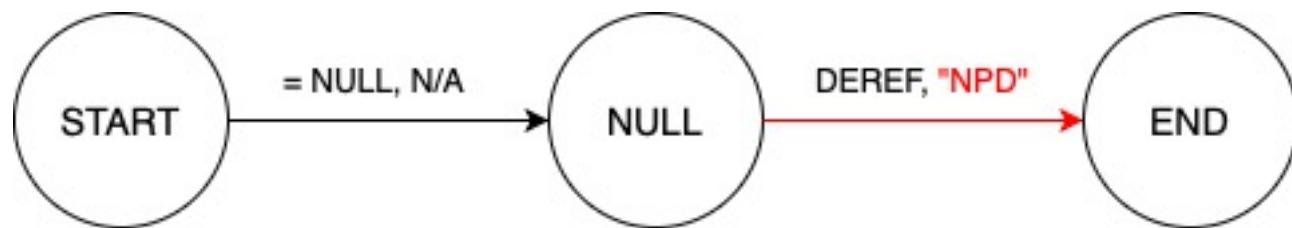
EXAMPLE: NULL POINTER DEREFERENCE FSM ABSTRACTION

A straightforward thought about the checker



EXAMPLE: NULL POINTER DEREFERENCE FSM ABSTRACTION

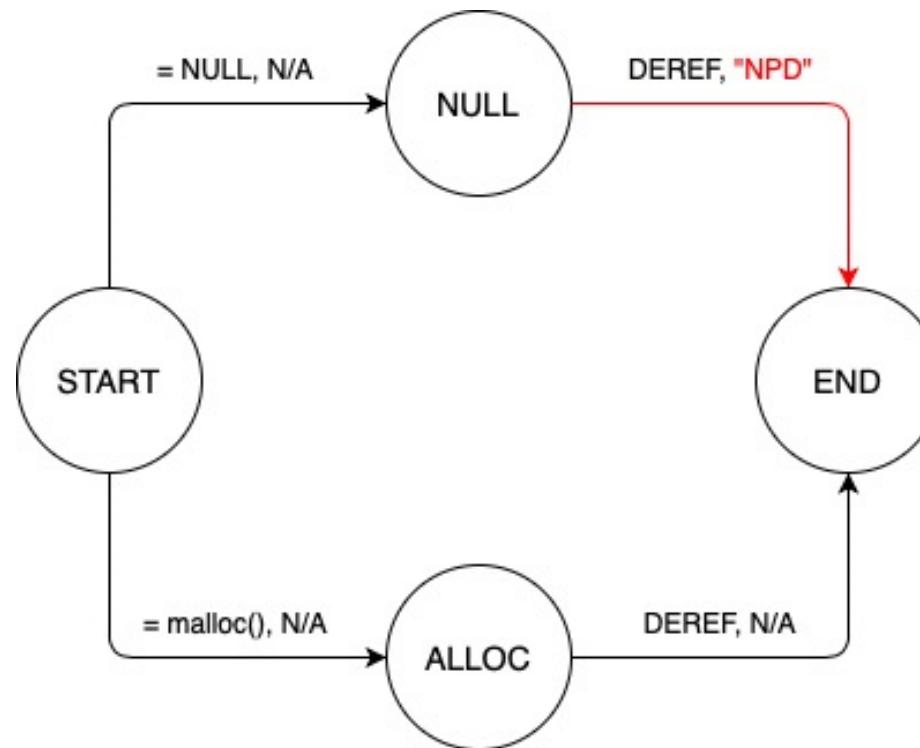
A straightforward thought about the checker



Think a little bit deeper: what's the 'good' behavior?

EXAMPLE: NULL POINTER DEREference FSM ABSTRACTION

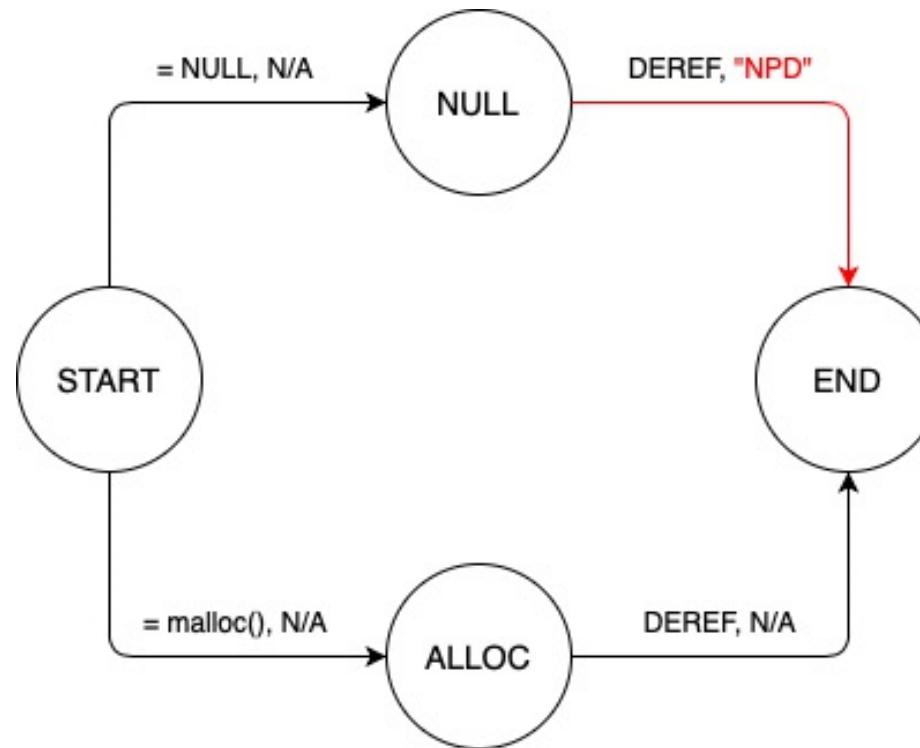
'good' behavior: malloced pointer should be fine to dereference



EXAMPLE: NULL POINTER DEREference FSM ABSTRACTION

4

It is not accurate: after malloc, the pointer may be free & malloc may fail

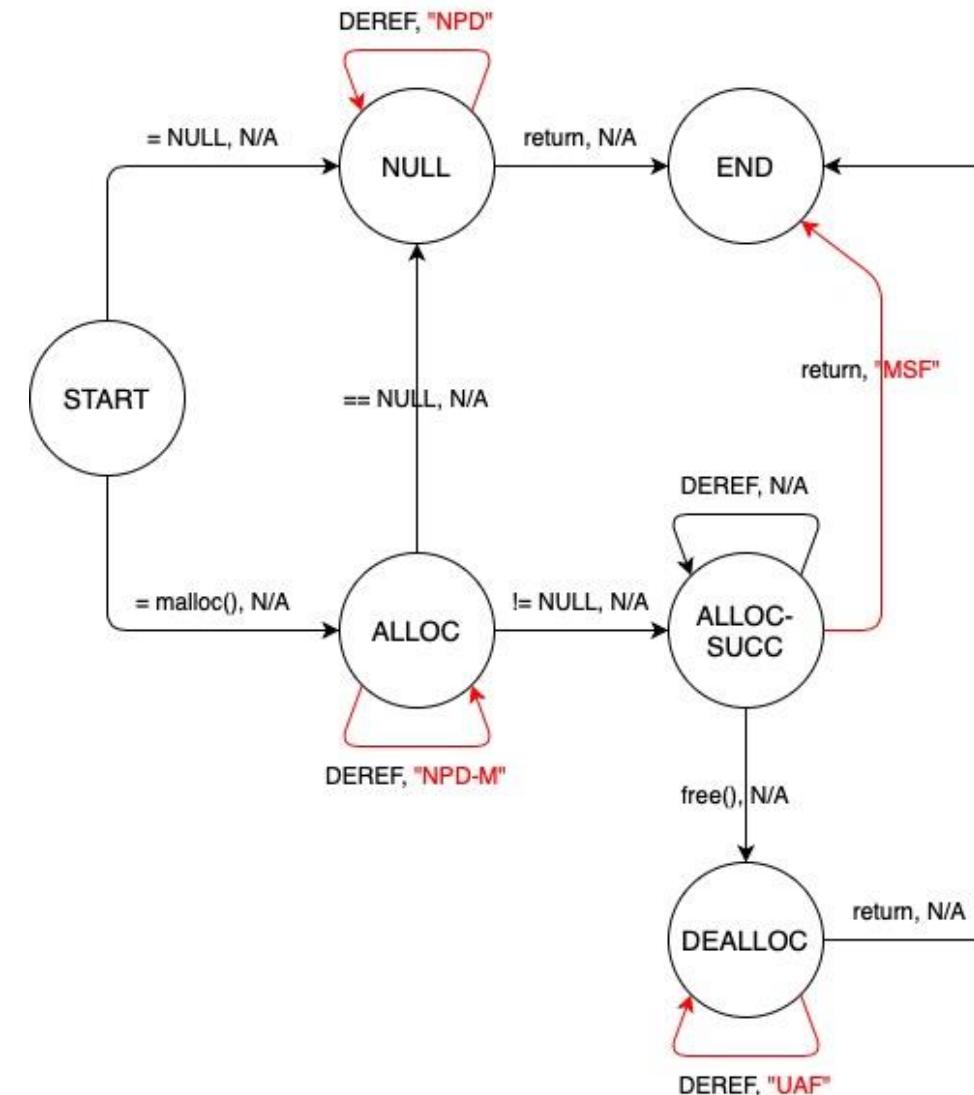


EXAMPLE: NULL POINTER DEREference FSM ABSTRACTION

Use After Freed

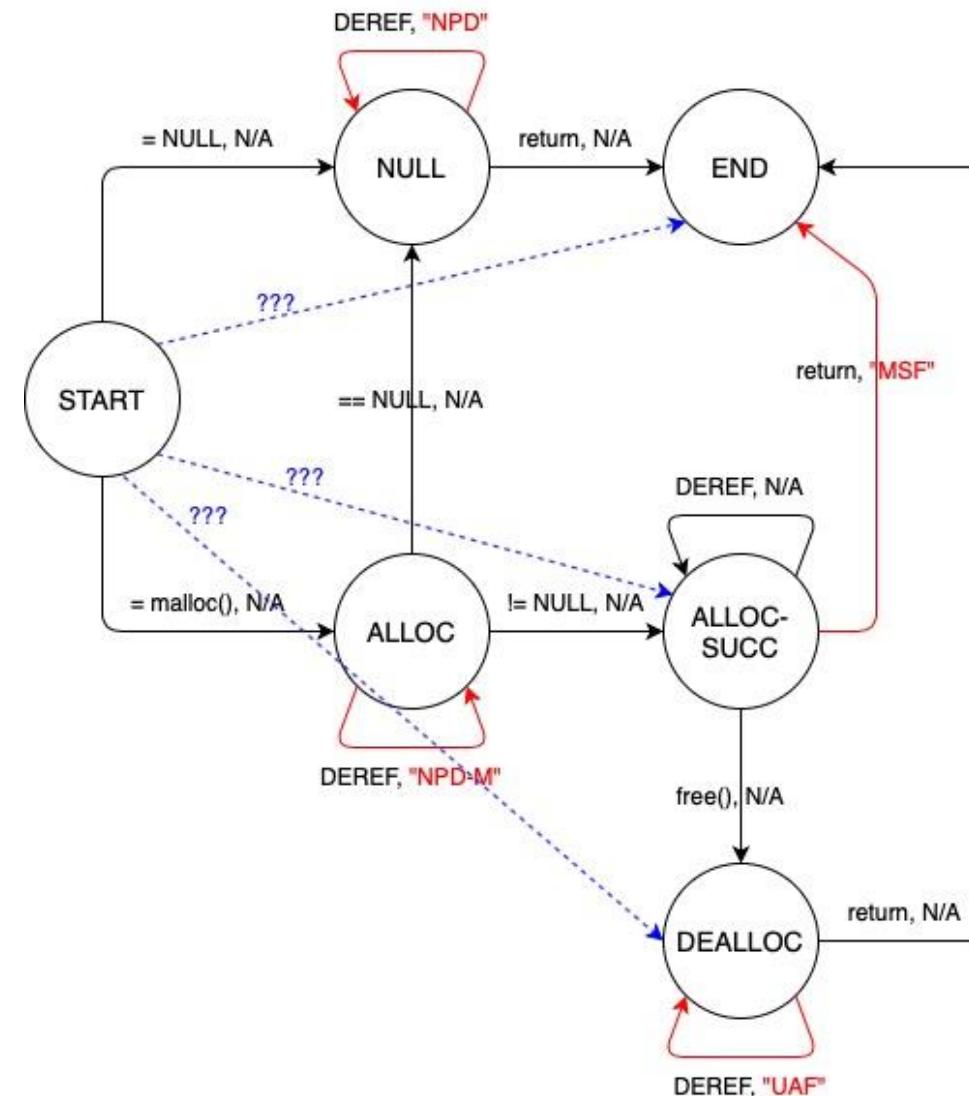
Null Pointer Dereference-Maybe

MiSSing Free



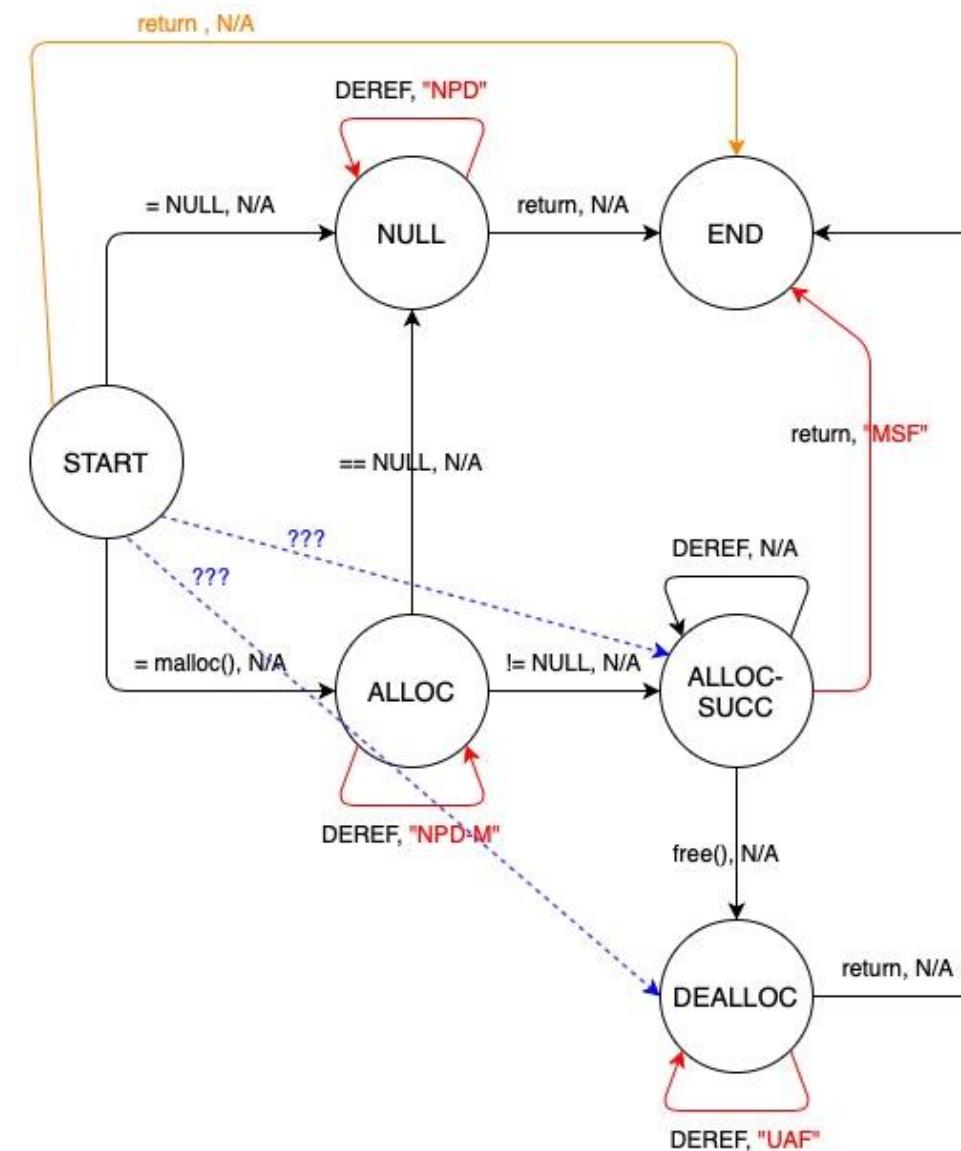
EXAMPLE: NULL POINTER DEREference FSM ABSTRACTION

Precise enough?



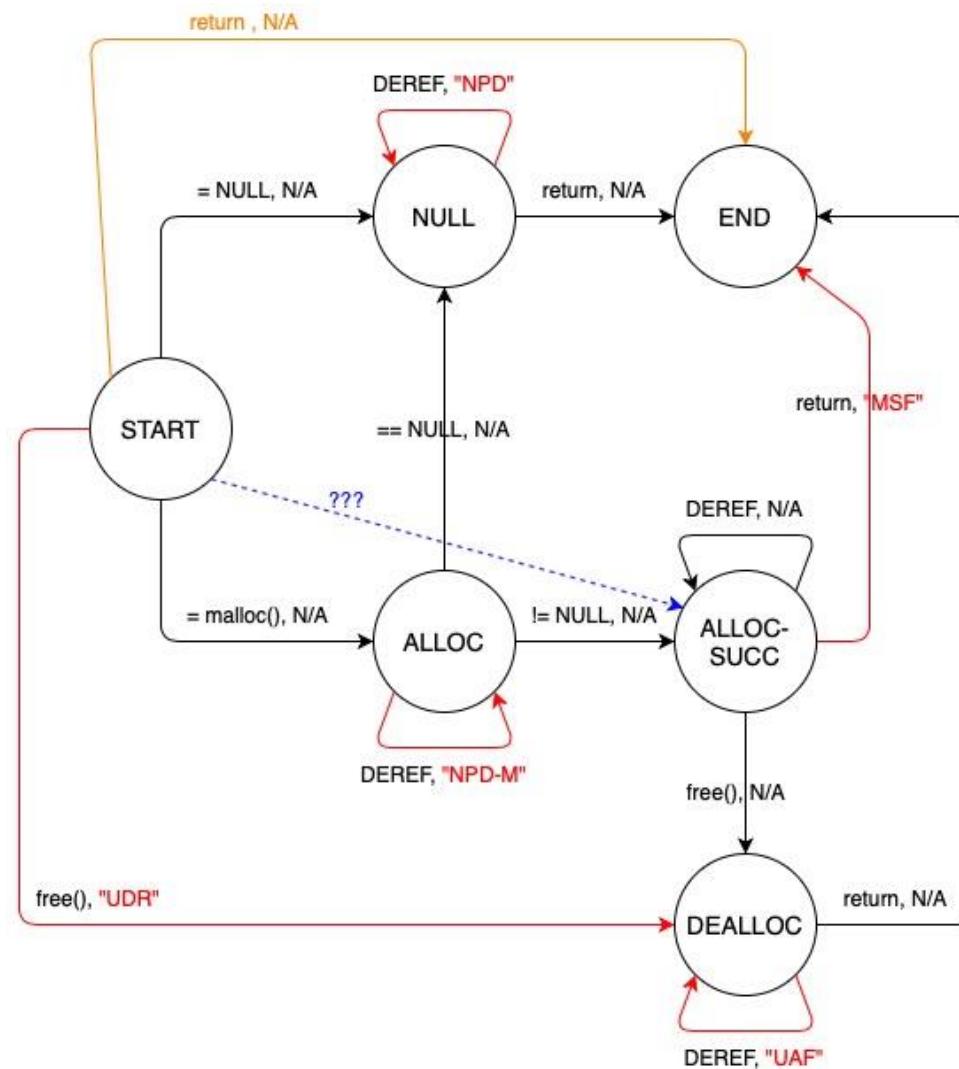
EXAMPLE: NULL POINTER DEREference FSM ABSTRACTION

No codes related



EXAMPLE: NULL POINTER DEREference FSM ABSTRACTION

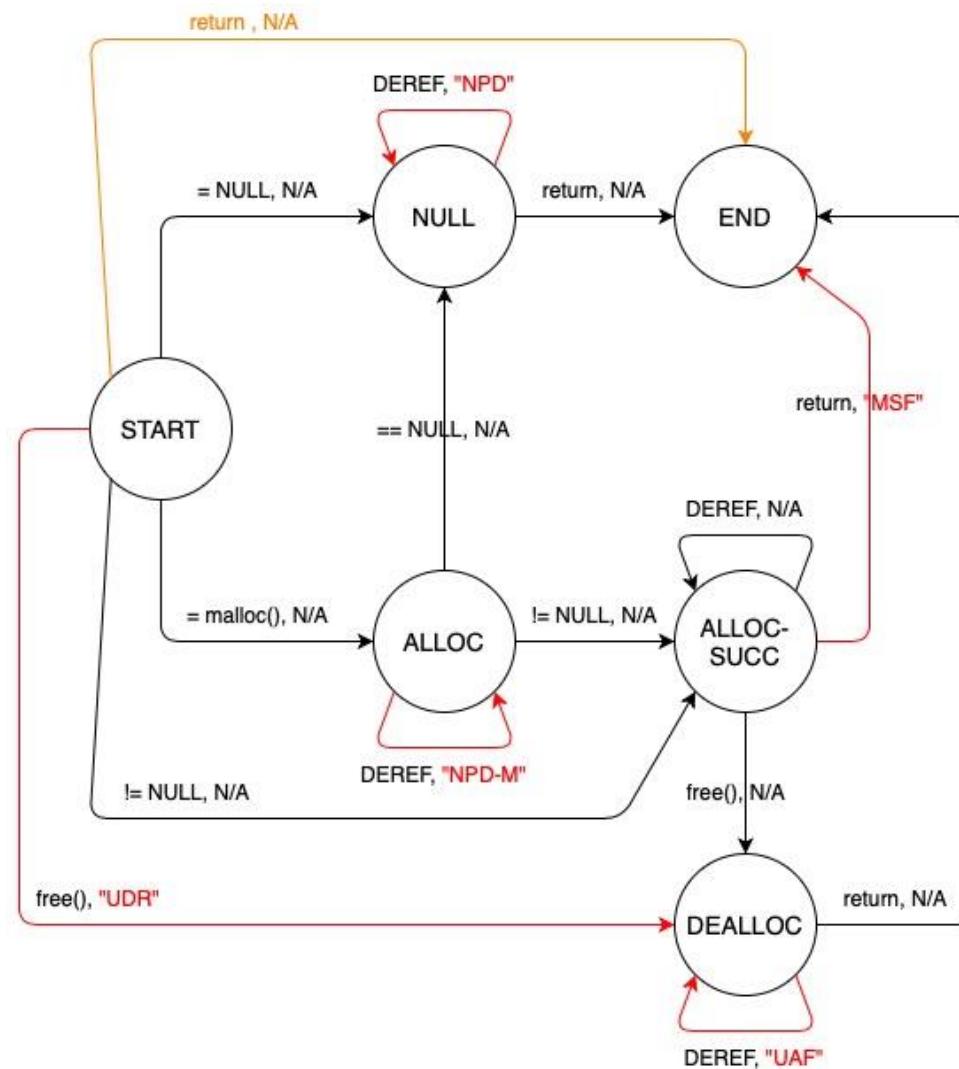
Use Dangling Reference



EXAMPLE: NULL POINTER DEREference FSM ABSTRACTION

A not-null check can avoid NPD as well

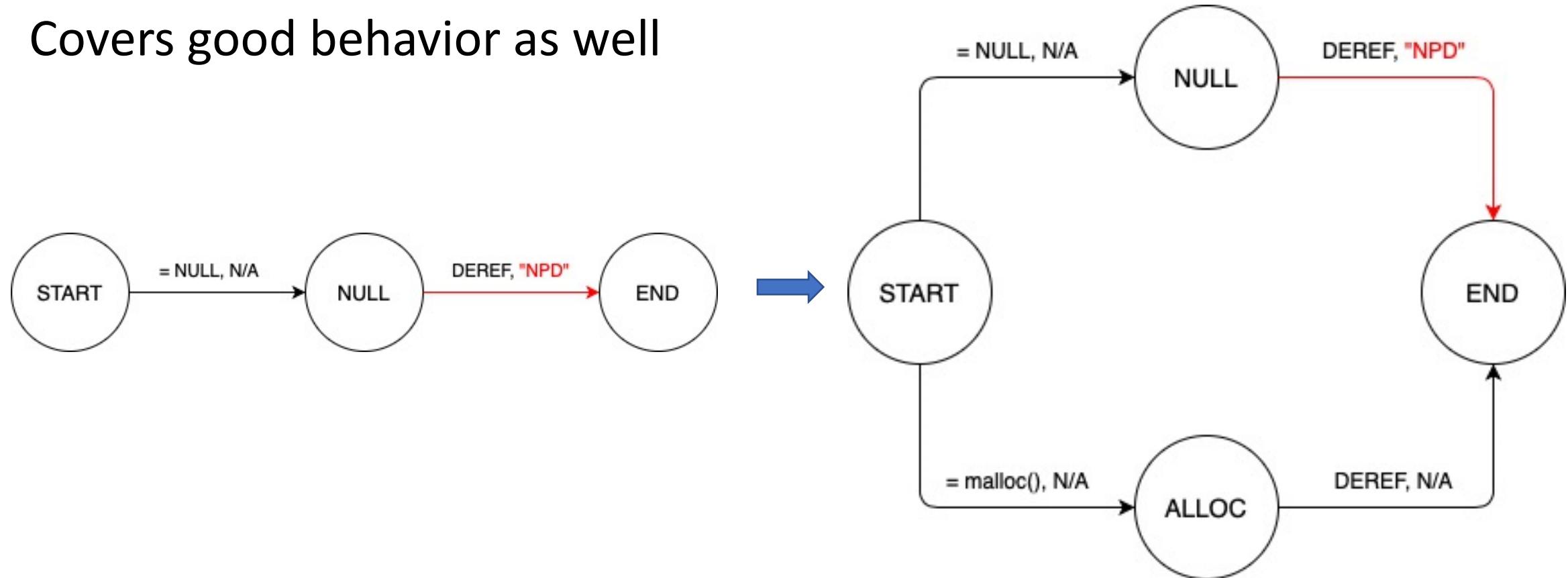
Consider library / SDK APIs





FSM ABSTRACTION BUILD UP

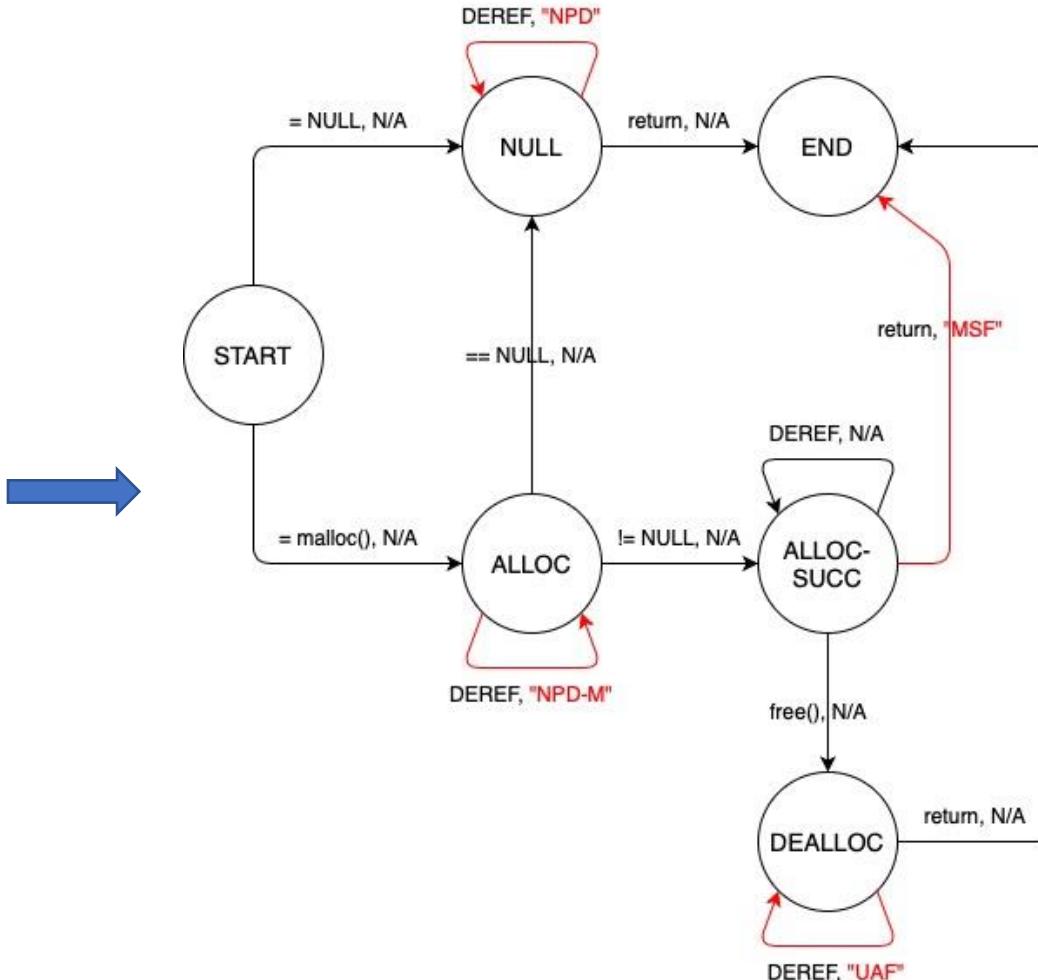
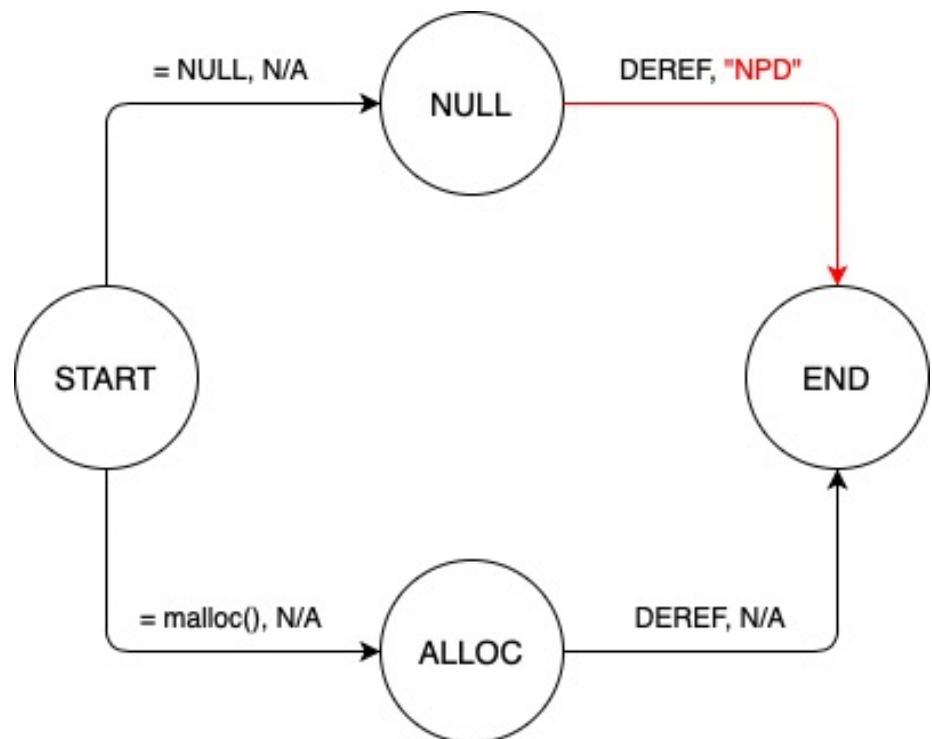
Covers good behavior as well





FSM ABSTRACTION BUILD UP

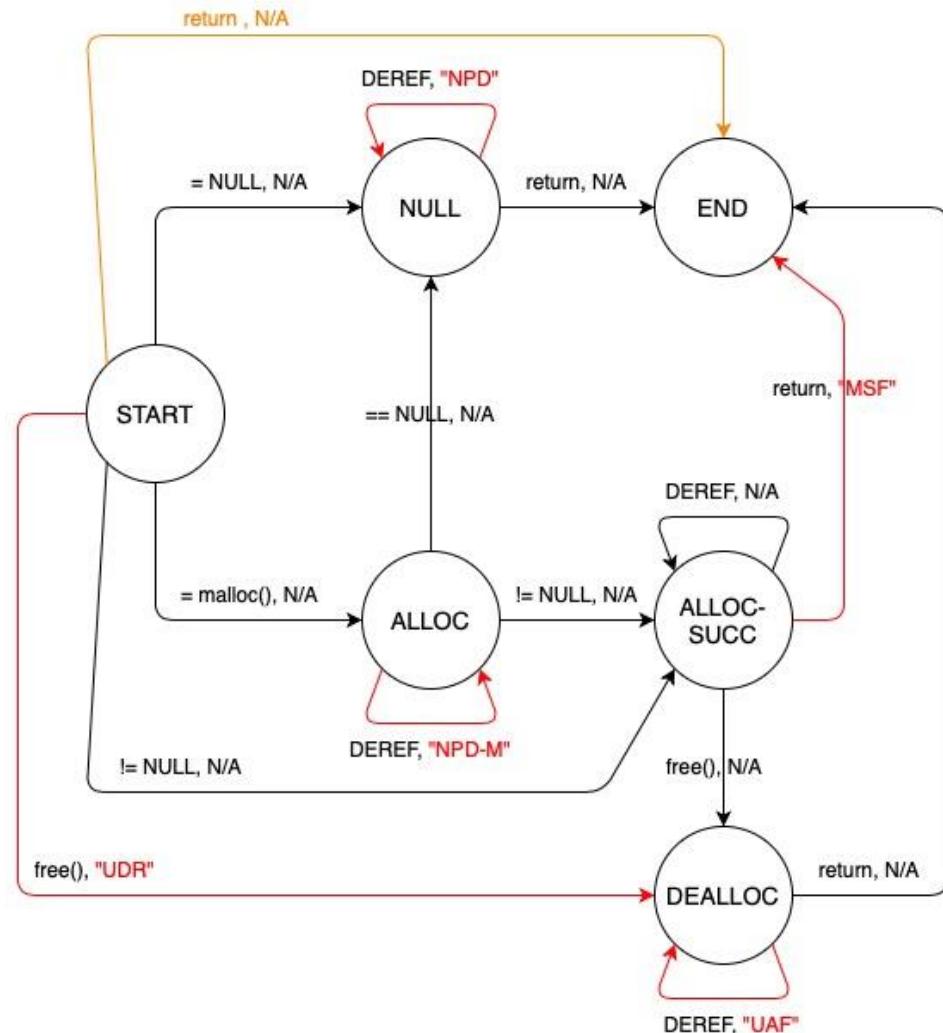
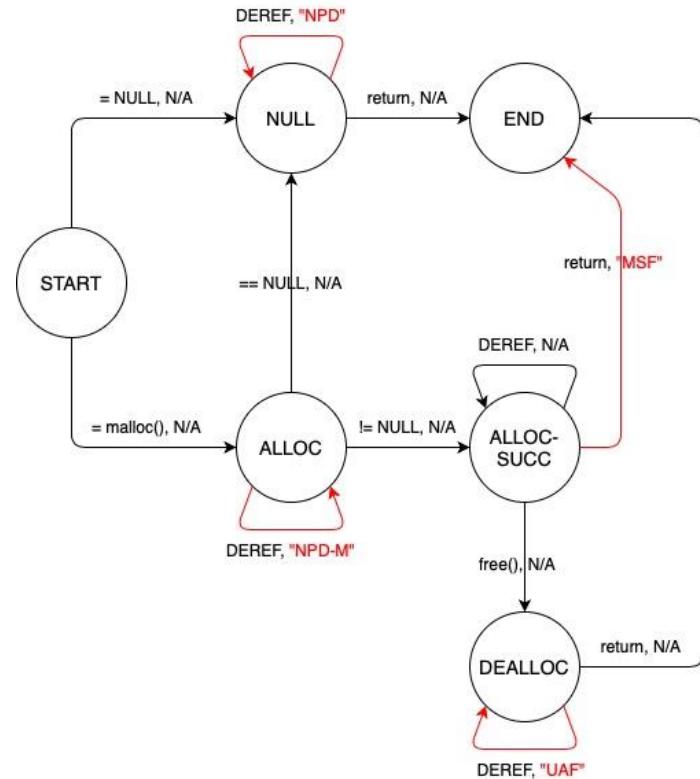
Considers the side effects





FSM ABSTRACTION BUILD UP

Tries to reach all other states from one





FSM ABSTRACTION BUILD UP

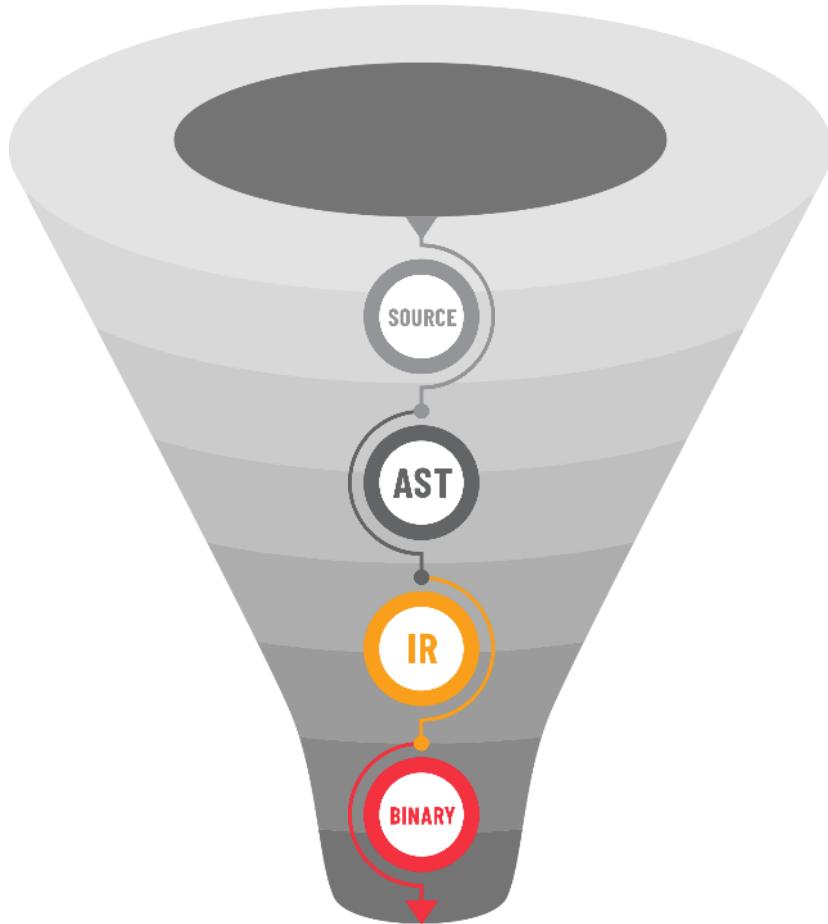
- ✓ Covers good behavior as well
- ✓ Considers the side effects
- ✓ Tries to reach all other states from one
- ✓ Iterates above until a fixed point

xcalscan PERFORMANCE EVALUATION

- ✓ Analyze based on SSA IRs
 - Context sensitive
 - Flow sensitive
 - Cross file
 - Cross language

- ✓ On demand analysis
 - Less time
 - Less memory

- ✓ Symbolic evaluation
 - User customizable rules



xcalscan PERFORMANCE EVALUATION



Analyze based on SSA IRs

- Context sensitive
- Flow sensitive
- Cross file
- Cross language



On demand analysis

- Less time
- Less memory



Symbolic evaluation

- User customizable rules

• ***Same analysis algorithm***

• ***As efficient as checkers for common vulnerabilities***

EVALUATION: TIME & MEMORY FOOTPRINT

| PROJECT | LOC | XCALSCAN* | | PINPOINT 1.5** | |
|----------------|------|-----------|--------------|----------------|--------------|
| | | Scan Time | Memory Usage | Scan Time | Memory Usage |
| MySQL 5.5.10 | 1M | 8m19s | 11.2GB | 1.5h* | 60GB* |
| OpenSSL 1.0.1f | 500K | 2m28s | 2.9GB | - | - |
| GDB 7.0a | 490K | 1m24s | 3.5GB | - | - |

C: ~3K LOC/s (65 projects, 16,438,505 LOC)

JAVA: ~0.5k LOC /s (9 projects, 736,828 LOC)

* XCALSCAN on CentOS7, 64GB Mem + 17GB Swap, Intel i7-9700 @ 3.00GHz (8 core)

** Source: PINPOINT PLDI2018 paper,
<https://www.cse.ust.hk/~charlesz/pinpoint.pdf>

EVALUATION: ACCURACY

| Juliet C | XCALSCAN | CLANG |
|-----------------------------------|----------|-------|
| True Positive (TP, Higher better) | 64% | 21% |
| False Positive (FP, Lower better) | 6% | 14% |

Benchmark

- Juliet C has 74699 cases, 118 categories.
- 42 supported, 26 of windows tests excluded

Version

- Clang 3.8.0-2

EVALUATION: ACCURACY CONTINUED

| PROJECT | RULE | XCALSCAN | | | | OTHER TOOL | | | |
|-------------------|------|-----------------|---------------|----------------|---------|-----------------|---------------|----------------|---------|
| | | Errors reported | True Positive | False Positive | FP Rate | Errors reported | True Positive | False Positive | FP Rate |
| MySQL 5.5.10 | NPD | 261 | 147 | 114 | 43% | 69 | 55 | 14 | 20% |
| | UAF | 40 | 22 | 18 | 45% | 19 | 12 | 7 | 36% |
| OpenSSL 1.0.1f | NPD | 65 | 46 | 19 | 29% | 48 | 42 | 6 | 12% |
| | UAF | 18 | 6 | 12 | 66% | 6 | 3 | 3 | 50% |
| GDB 7.0a | NPD | 88 | 50 | 38 | 43% | 12 | 11 | 1 | 8% |
| | UAF | 15 | 4 | 11 | 73% | 5 | 1 | 4 | 80% |

EVALUATION: CROSS PROCEDURE

| Juliet C | XCALSCAN | XCALSCAN (cross procedure feature disabled) |
|-----------------------------------|----------|--|
| True Positive (TP, Higher better) | 64% | 36% |
| False Positive (FP, Lower better) | 6% | 28% |

- ✓ Benefit of cross procedure analysis
- Higher True Positive rate
 - Lower False Positive rate

EXAMPLE: INJECTION – CROSS PROCEDURE

```
public class MyClass {  
    public String myCmd(HttpServletRequest request) {  
        ...  
        String param = request.getParameter("taintedParam");  
        String cmd = ... + param;  
        ...  
        return cmd;  
    }  
    public void myFunc(HttpServletRequest request) {  
        ...  
        String cmd = myCmd(request);  
        Runtime r = Runtime.getRuntime();  
        Process p = r.exec(cmd);  
        ...  
    }  
}
```

EXAMPLE: INJECTION – CROSS PROCEDURE

```
public class MyClass {
    public String myCmd(HttpServletRequest request) {
        ...
        String param = request.getParameter("taintedParam");
        String cmd = ... + param;
        ...
        return cmd;
    }
    public void myFunc(HttpServletRequest request) {
        ...
        String cmd = myCmd(request);
        Runtime r = Runtime.getRuntime();
        Process p = r.exec(cmd);
        ...
    }
}
```

Cross procedure by nature

#1 Recognize tainted variable

```
public interface ServletRequest {
    default public String getParameter(String var) {
        SEE.SideEffect(SEE.SetAttr(See.FuncRet(), "tainted"));
        ...
    }
}
```

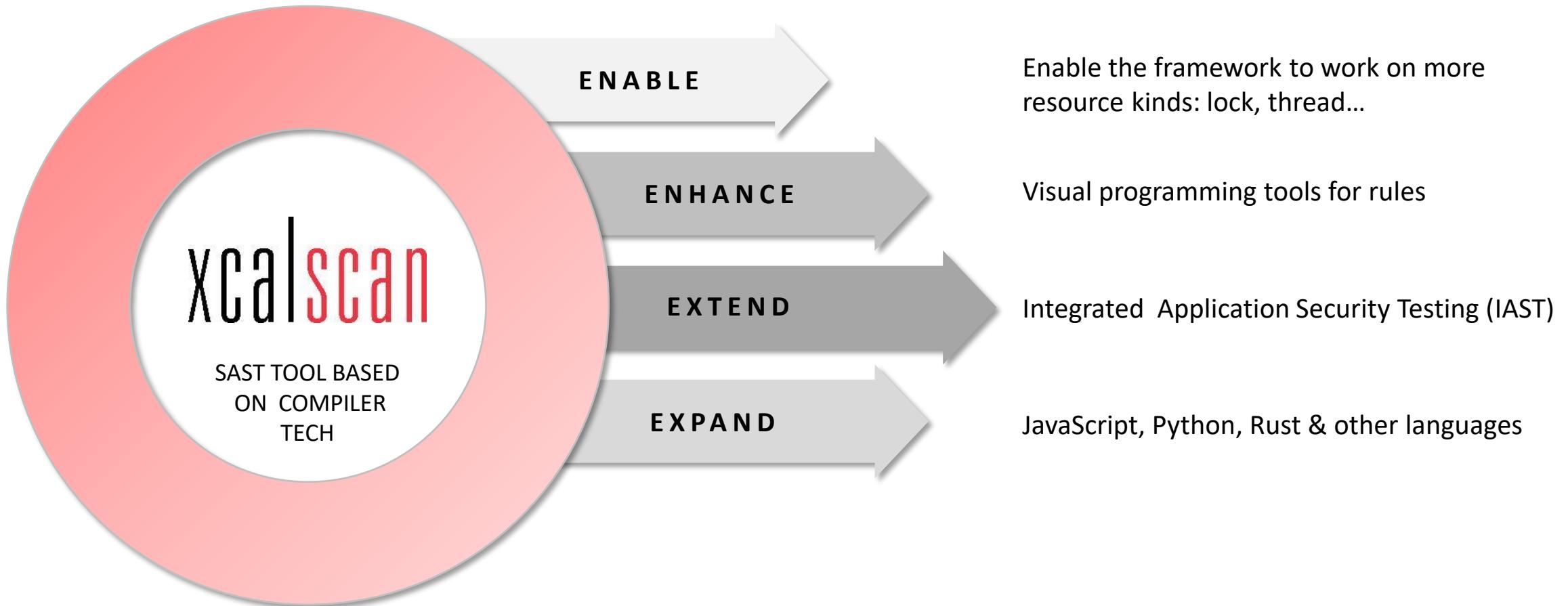
#3 Add check point

```
public class Runtime{
    public Process exec(String command) {
        SEE.Assert(SEE.Attr(SEE.Arg(1)) != "tainted", "tainted cmd");
        ...
    }
}
```

SUMMARY

- 1 Business Logic verification is critical
- 2 Symbolic evaluation enables customizability
- 3 Evaluate SAST by speed, memory consumption, accuracy & cross procedural capability

XCALIBYTE'S FUTURE DIRECTIONS



Q&A

The background image shows a panoramic view of a city skyline during sunset or sunrise. The sky is filled with dramatic, colorful clouds in shades of orange, yellow, and blue. In the foreground, the silhouette of a person is sitting on the edge of a tall building's roof, facing away from the camera towards the horizon. The city buildings are dark against the bright sky, with some lights visible on the windows of taller structures.

鉴释

xcalibyte

马骏 (良斌)

阿里云智能 C/C++ 编译器技术主管



拥有多年编译器开发和性能优化经验，熟悉GCC LLVM的Middle&Back End. 先后在阿里巴巴编译器团队 负责集团C/C++ workload的性能优化，推动AutoFDO优化的上线。参与GCC/LLVM社区C++20标准中协程部分的开发，并在阿里推广和大规模落地。

主办方：

Boolan
高端 IT 咨询与教育平台

C++20协程与应用

阿里云-程序语言与编译器

- 1、协程
- 2、C++20协程
- 3、编译器相关工作
- 4、future_lite协程库
- 5、业务落地
- 6、未来

1、协程

什么是协程？

- 协程是一种程序组件，是由子例程（过程、函数、例程、方法、子程序）的概念泛化而来的，子例程只有一个入口点且只返回一次，而协程允许多个入口点，可以在指定位置挂起和恢复执行。
 - ✓ 协程在控制离开时暂停执行，当控制再次进入时只能从离开的位置继续执行。
 - ✓ 协程的本地数据在后续调用中持久化。

为什么要用协程？

- VS 多线程
 - ✓ uthread：极高的执行&切换效率；
- VS 异步回调；
 - ✓ 统一的同步编程模式， 避免callback hell

协程分类：

libeasy, libco, boost.coroutine, go coroutine, C# await...

- 控制传递（Control-transfer）机制：对称/非对称（return-to-caller）
- 是否作为语言的第一类（First-class）对象提供：语言特性&编译器支持
- 栈式（Stackful）/无栈（Stackless）构造：是否在内部的嵌套调用中挂起（性能&挂起位置）

2、C++20协程



- C++20语言的重要的特性（Big Four - concept,ranges,module,coroutine）；
- C++20协程：对称、语言第一类支持、无栈
- 三个关键字：co_await、co_yield、co_return

一个例子：

```
Resumable func () {
    std::cout << "Hello" << std::endl;
    co_await std::experimental::suspend_always();
    std::cout << "Coroutine" << std::endl;
}
```

一个例子：

```
Resumable func () {
    std::cout << "Hello" << std::endl;
    co_await std::experimental::suspend_always();
    std::cout << "Coroutine" << std::endl;
}
```

```
$ clang++ -std=c++17 -O0 -c x.cc -fcoroutines-ts -stdlib=libc++
x.cc:10:11: error: this function cannot be a coroutine: 'std::experimental::coroutines_v1::coroutine_traits<Resumable>' has no
      member named 'promise_type'
Resumable func() {
^
1 error generated.
```

如何实现Resumable?

- 协程句柄 (coroutine handle) & 协程对象;
- promise_type结构;

编译器第一次改造：

```
Resumable func(args...){  
    Frame *frame_ = operator new(std::size_t size);  
    Promise_type promise; // 1  
    coroutine_handle *handle_ = coro_handle::from_promise(&promise);  
    Resumable ret = promise.get_return_object(); // 2  
    co_await promise.initial_suspend();  
  
    try {  
        // co-routine body  
    } catch(...) {  
        promise.unhandled_exception();  
    }  
  
    final_suspend:  
    co_await promise.final_suspend();  
  
    return ret;  
}
```

- 1、编译器选择promise_type的两个方法：
 - Resumable里定义promise_type类型
 - 特例化coroutine_traits<Resumable, Args...>类型
 - 2、Coroutine_handle:和协程对象一一对应，可以恢复/销毁协程。在experimental/coroutine中实现。
 - 3、返回Resumable对象
- 程序员实现：promise_type、Resumable

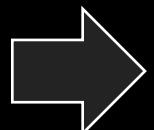
实现Resumable:

```
class Resumable {
public:
    struct promise_type;
    bool resume() {
        if (!handle_.done())
            handle_.resume();
        return !handle_.done();
    }
    coro_handle handle_; // 协程句柄
};

struct Resumable::promise_type {
    Resumable get_return_object() {/**/}
    auto initial_suspend() {/**/}
    auto final_suspend() {/**/}
    void return_void() {}
    void unhandled_exception();
};
```

编译器第一次改造：

```
Resumable func () {
    std::cout << "Hello" << std::endl;
    co_await std::experimental::suspend_always();
    std::cout << "Coroutine" << std::endl;
}
```



```
Resumable func(args...) {
    Frame *frame_ = operator new(std::size_t size);
    Promise_type promise;
    coroutine_handle *handle_ = coro_handle::from_promise(&promise);
    Resumable ret = promise.get_return_object();

    co_await promise.initial_suspend();

    try {

        // co-routine body
        std::cout << "Hello" << std::endl;
        co_await std::experimental::suspend_always();
        std::cout << "Coroutine" << std::endl;

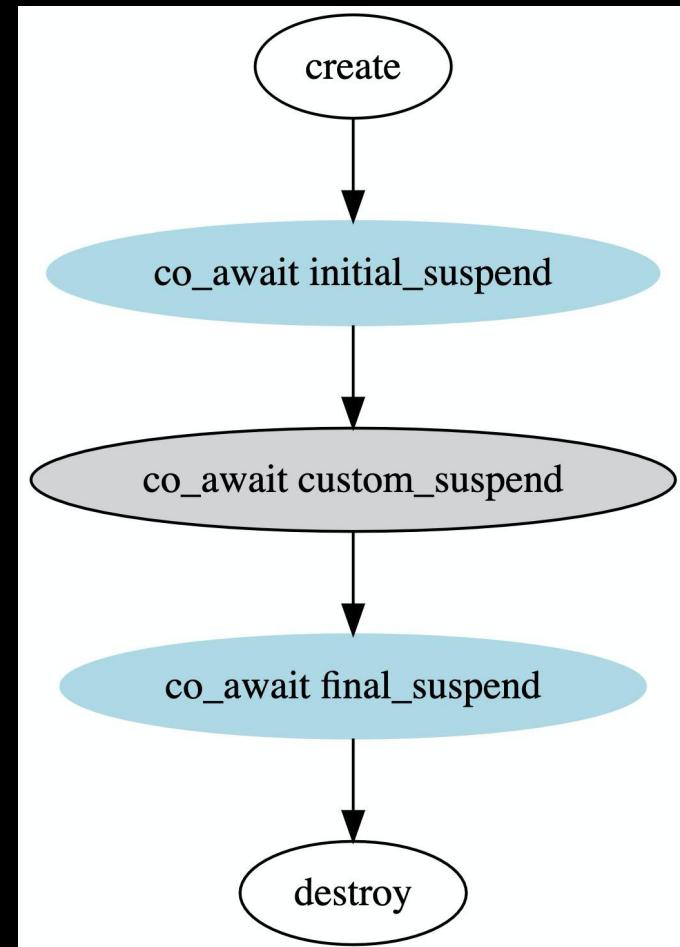
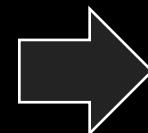
    } catch(...) {
        promise.unhandled_exception();
    }

    final_suspend:
    co_await promise.final_suspend();

    return ret;
}
```

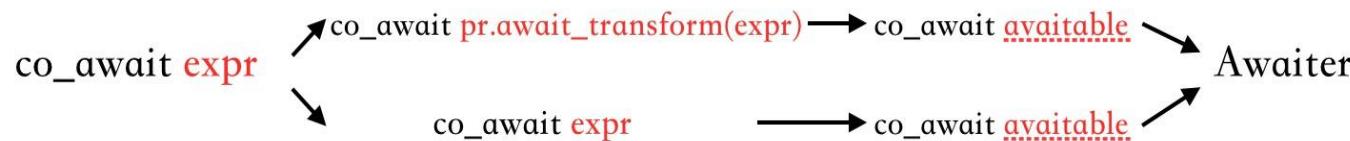
编译器第一次改造：

```
Resumable func () {
    std::cout << "Hello" << std::endl;
    co_await std::experimental::suspend_always();
    std::cout << "Coroutine" << std::endl;
}
```



Awaiter

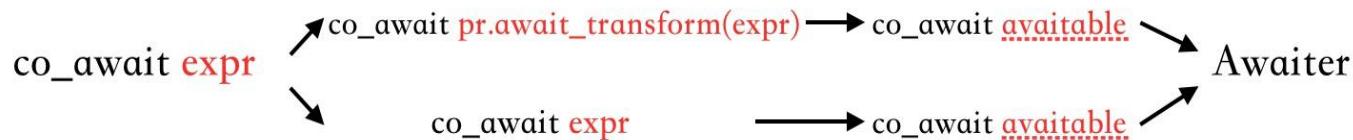
```
co_await std::experimental::suspend_always();
```



```
struct Awaiter {  
    bool await_ready() {...}  
    auto await_suspend(coroutine_handle<>) {...}  
    auto await_resume() {...}  
};
```

Awaiter

```
co_await std::experimental::suspend_always();
```



```
struct Awaitee {
    bool await_ready() {...}
    auto await_suspend(coroutine_handle<>) {...}
    auto await_resume() {...}
};
```

```
//1: if await_suspend returns void
if (not a.await_ready()) {
    try {
        a.await_suspend(coroutine_handle);
        return_to_the_caller();
    } catch (...) {
        exception = std::current_exception();
        goto resume_point;
    }
    //endif
}

resume_point:
if(exception)
    std::rethrow_exception(exception);
return a.await_resume();
```

```
//2: if await_suspend returns bool
if (not a.await_ready()) {
    bool await_suspend_result;
    try {
        await_suspend_result = a.await_suspend(coroutine_handle);
    } catch (...) {
        exception = std::current_exception();
        goto resume_point;
    }
    if (not await_suspend_result)
        goto resume_point;
    return_to_the_caller();
    //endif
}

resume_point:
if(exception)
    std::rethrow_exception(exception);
return a.await_resume();
```

```
//3: if await_suspend returns another coroutine_handle
if (not a.await_ready()) {
    decltype(a.await_suspend(std::declval<coro_handle_t>())) another_coro_handle;
    try {
        another_coro_handle = a.await_suspend(coroutine_handle);
    } catch (...) {
        exception = std::current_exception();
        goto resume_point;
    }
}
another_coro_handle.resume();
return_to_the_caller();
//endif

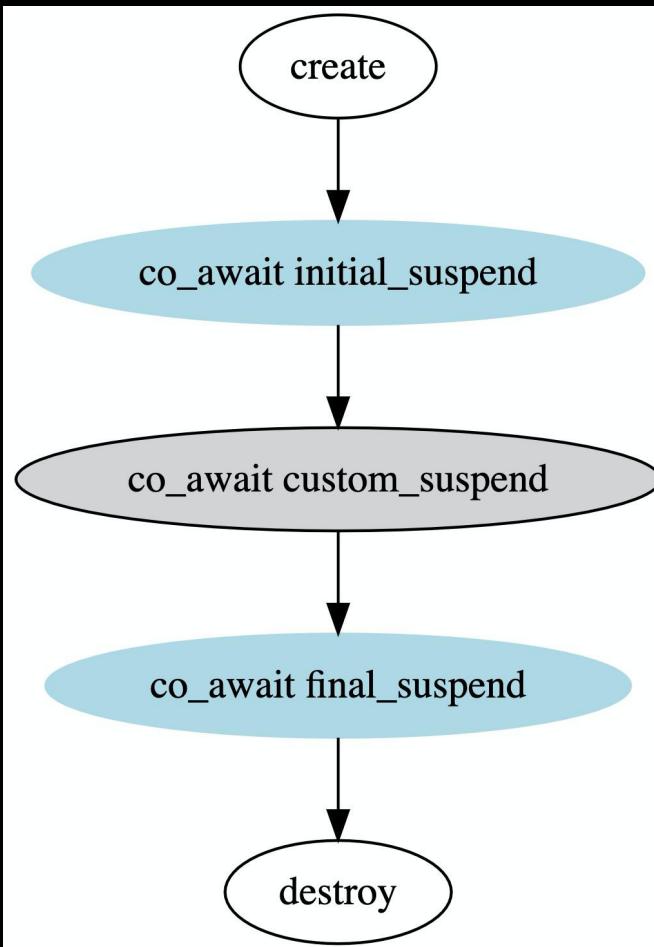
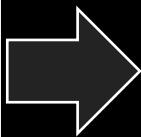
resume_point:
if(exception)
    std::rethrow_exception(exception);
return a.await_resume();
```



服务 商

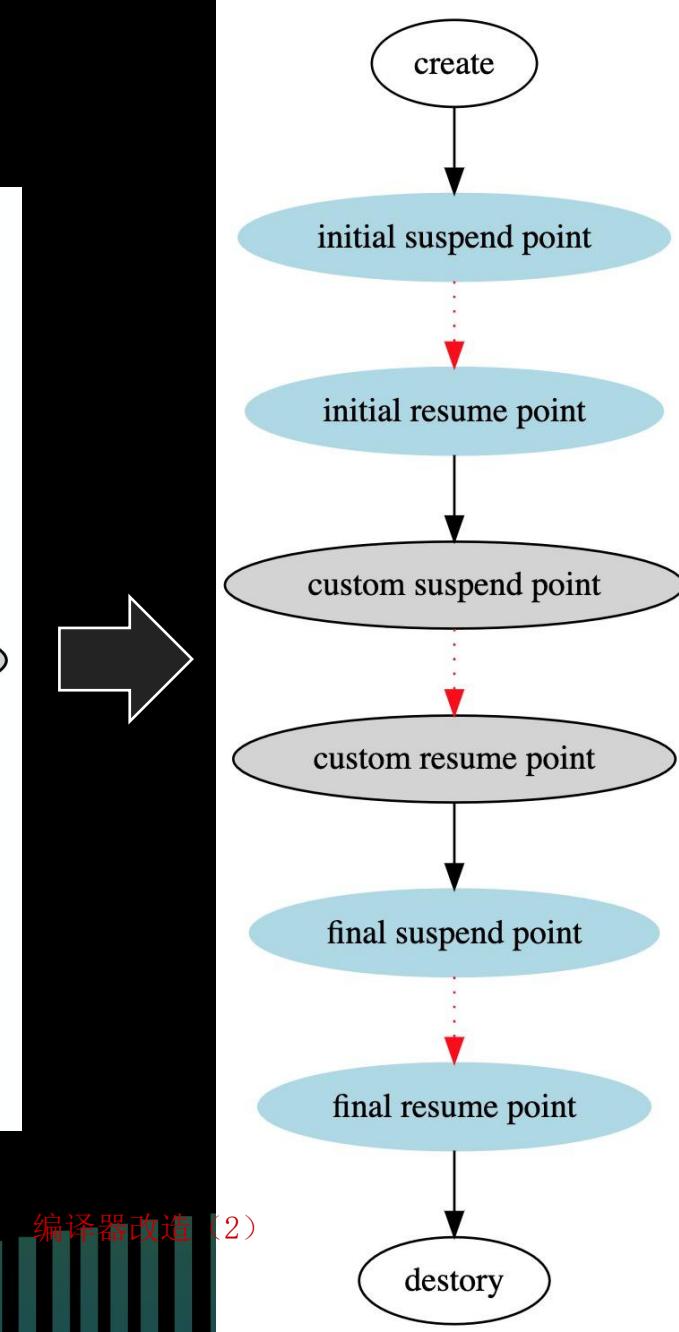
编译器第二次改造：

```
Resumable func () {
    std::cout << "Hello" << std::endl;
    co_await std::experimental::suspend_always();
    std::cout << "Coroutine" << std::endl;
}
```



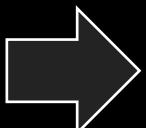
编译器改造 (1)

编译器改造 (2)



编译器第三次改造：

```
Resumable func () {
    std::cout << "Hello" << std::endl;
    co_await std::experimental::suspend_always();
    std::cout << "Coroutine" << std::endl;
}
```



```
// ramp function
Resumable func.ramp(args...){
    Frame *frame_ = operator new(std::size_t size);
    {
        // construct coroutine frame, there are fields such as:
        promise_type obj
        args...
        local variables
        // also there are following fields which relate to coroutine suspend/resume
        frame_.index = 0
        frame_.resume = func.resume;
        frame_.destroy = func.destroy
    }
    coroutine_handle r = frame_->promise.get_return_object();
    r.resume() // eventually call func.resume()
    return Resumable(r);
}
```

```
// resume function
void func.resume(Frame *frame_) {
    switch(frame_.index) {
        case 0:
            // ...
            frame_.index = 1;
            initial suspend point;

        case 1:
            // initial resume point
            // ...
            frame_.index = 2;
            custom suspend point;

        case 2:
            // custom resume point
            // ...
            frame_.index = 3;
            final suspend point;

        // case ...:

        case N:
            // final resume point
            frame_.destroy() // func.destroy()
    }
}
```

```
// destroy function
void func.destroy(Frame *frame_) {
    // release coroutine's resources and destroy the coroutine object
}
```

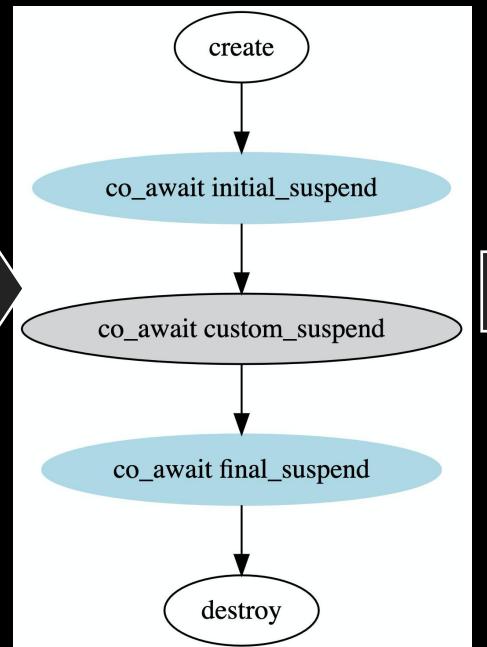
1、func.ramp:负责申请协程帧，保存协程参数、局部变量、promise obj、index、resume函数指针、destroy函数指针等

2、func.resume:简单状态机，根据frame_.index，选择恢复点。

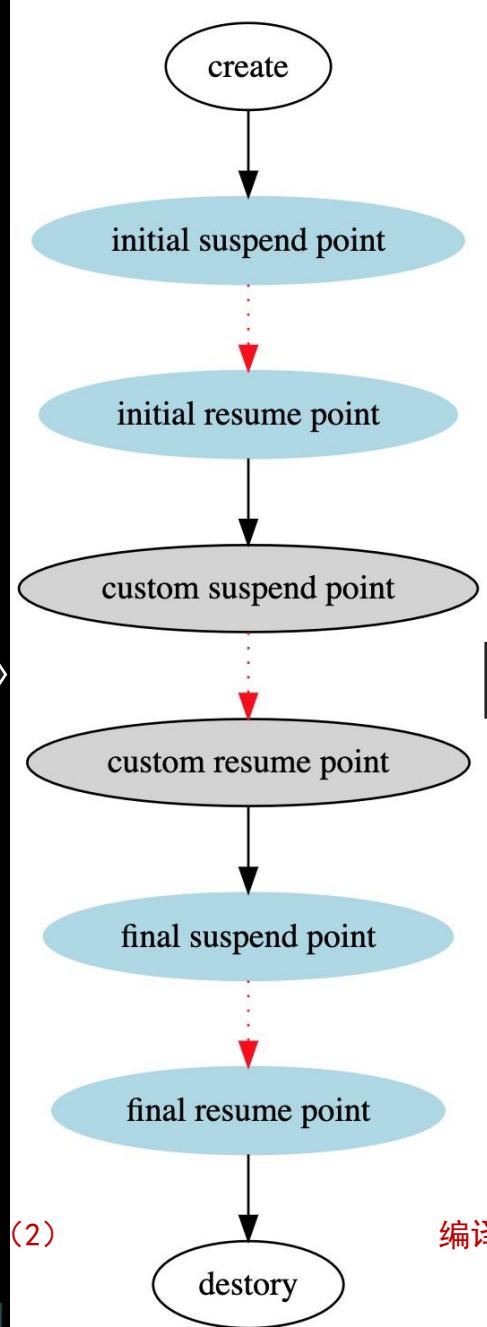
3、func.destroy: 回收资源

编译器的**三次改造**:

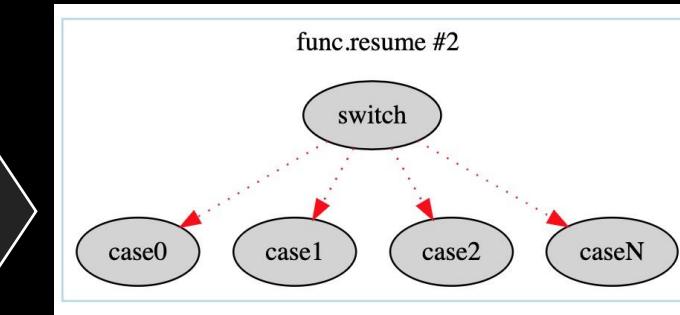
```
Resumable func () {
    std::cout << "Hello" << std::endl;
    co_await std::experimental::suspend_always();
    std::cout << "Coroutine" << std::endl;
}
```



编译器改造 (1)



编译器改造 (2)



编译器改造 (3)

实例执行：

```
#include "resumable.h"

Resumable func() {
    std::cout << "Hello" << std::endl;
    co_await std::experimental::suspend_always();
    std::cout << "Coroutine" << std::endl;
}

int main() {
    Resumable res = func();
    while(res.resume())
        ;
    return 0;
}
/*
Hello
Coroutine
*/
```

3、编译器相关工作

GCC VS LLVM实现策略

- 协程函数展开
- 协程函数拆分
- Inline vs Split
- GCC在AST生成IR时就将协程函数拆开，优化策略为对已经生成好的协程函数进行分析和优化（Inline）
- LLVM则是基于IR做协程函数的拆分，利用已有优化对协程函数优化后，再拆分（Split）并优化

阿里巴巴从2019年10月开始C++协程相关工作

➤ GCC

- ✓ 与社区合作进行协程的支持。
- ✓ GCC-10将是第一个支持C++协程特性的GCC编译器。
- ✓ 仅支持，无优化。

➤ LLVM

- ✓ 与Clang/LLVM社区合作完善C++协程。
- ✓ Clang/LLVM-11版本将具有成熟的协程特性支持，集团内Clang/LLVM-8稳定支持协程。
- ✓ 改善&优化：协程逃逸分析和CoroElide优化，协程帧优化（Frame reduction），改进协程调试信息、use-after-free、尾调用优化等。

4、future_lite协程库

现状

- 缺少STL标准库支持，C++23可能会支持；
- 典型第三方协程库
 - ✓ 开源的cppcoro库
 - ✓ 阿里巴巴集团future_lite库（借鉴了facebook/folly库）

future_lite库

- Lazy<T>组件
- future/promise组件
- 协程执行器-Executor
- 批量执行组件
- Generator/AsyncGenerator组件
- 异步Mutex/ConditionVariable 组件
- Barrier、Latch、Semaphore、SharedMutex、SpinLock、SharedLock等异步组件

future_lite库应用实例：

```
// 协程函数getValue
template<typename T>
Lazy<T> getValue(T x) {

    struct ValueAwaiter {
        T value;
        ValueAwaiter(T v) : value(v) {}
        bool await_ready() { return false; }
        void await_suspend(std::experimental::coroutine_handle<continuation> continuation) noexcept {
            std::thread([c = continuation] mutable { c.resume(); }).detach();
        }
        T await_resume() noexcept { return value; }
    };

    co_return co_await ValueAwaiter(x);
}

// 协程函数task2, 默认T = void
Lazy<> task2() {
    auto t = getValue(10);
    auto x = co_await t; // 恢复协程getValue/t的执行
    assert(x == 10);
}

// 普通函数func
void func() {
    SimpleExecutor exec;
    auto t1 = task1(10);
    auto x = syncAwait(t1.via(&exec)); // 指定执行器exec执行协程task1/t
    assert(x == 10);
}
```

```
Generator<std::uint64_t> fibonacci() {
    std::uint64_t a = 0, b = 1;
    while (true) {
        co_yield b; // yield a value
        auto tmp = a;
        a = b;
        b += tmp;
    }
};

for (auto i : fibonacci()) {
    if (i > 100'000) break;
    std::cout << i << std::endl;
}
```

future_lite库性能：

- 有栈协程：对比C++20协程和libeasy协程
注：libeasy阿里巴巴集团内部协程库
- 无栈协程：future_lite库和cppcoro库的性能

| | | future_lite | cppcoro | libeasy |
|----------------|-----|-------------|----------|----------|
| 场景1：单协程重复执行 | 场景1 | 181 ns | 271 ns | < 100 us |
| 场景2：线程池+海量协程任务 | 场景2 | 217 ms | 458.8 ms | 272 ms |
| 场景3：协程调用链 | 场景3 | 252 ns | 280 ns | — |
| 场景4：批量执行接口 | 场景4 | 219.3 ms | 302.6 ms | — |

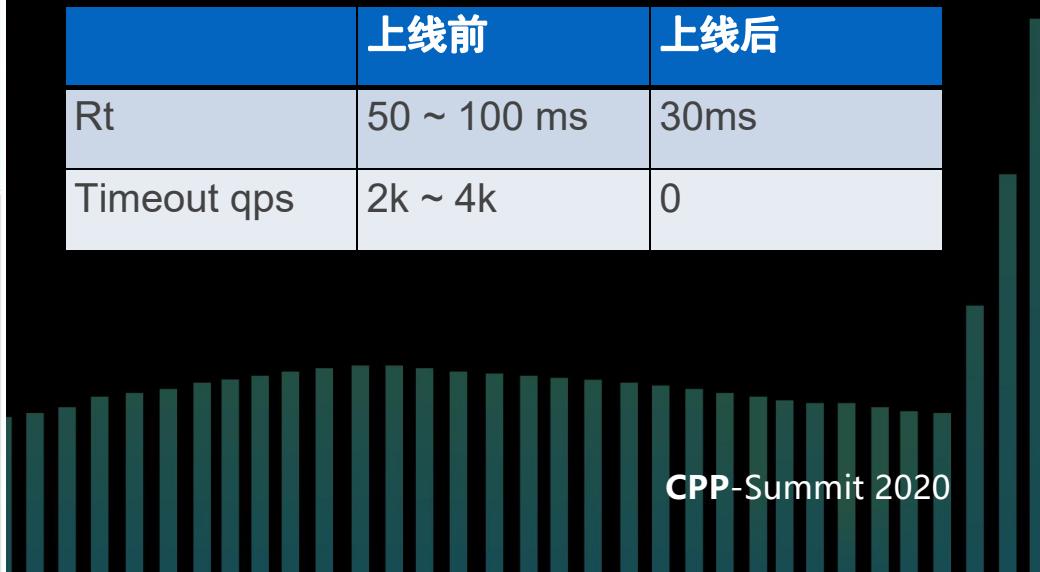
5、业务落地

阿里集团搜索、交互式分析等业务：

- 全链路改造
 - ✓ 查询、计算、存储
 - ✓ 方案：C++20协程 + 自研调度器 + 异步io改造。



| | 上线前 | 上线后 |
|-------------|-------------|------|
| Rt | 50 ~ 100 ms | 30ms |
| Timeout qps | 2k ~ 4k | 0 |



6、未来

➤ LLVM

- ✓ Add missing debugInfo
- ✓ IPO CoroElide
- ✓ Safe stack&frame check

➤ future_Lite

- ✓ 开源
- ✓ 有栈&无栈、同步&异步混合模式
- ✓ 支持更多IO/Network接口



奥运会全球指定云服务商

潘爱民

指令集公司CEO，著名操作系统专家



潘爱民，于2018年创办杭州指令集智能科技有限公司，快速崛起为国内物联网操作系统领域的领军企业。在此之前，潘爱民曾历任阿里YunOS、阿里安全、飞猪、阿里业务平台的首席架构师，也是商业操作系统的率先提出者。曾任盛大创新科学院首席科学家，微软亚洲研究院研究员。曾在北京大学和清华大学从事教学和研究，著作或译作多部计算机经典书籍或教材，影响一代程序员，是计算机工业界的领军人物。

主办方：

Boolan
高端IT咨询与教育平台

C++ Summit 2020

潘爱民



ISYSCORE

杭州指令集智能科技

物联网操作系统的 架构设计



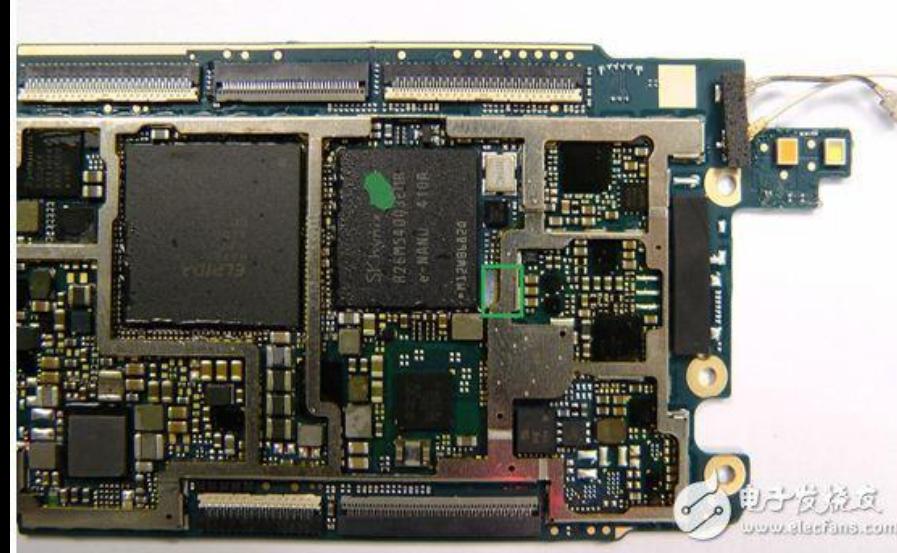
操作系统的硬件环境 —— PC



- 核心计算环境：CPU、RAM、HDisk、.....
- 外设：键盘、鼠标、打印机、.....
- 操作系统：Linux、Windows、MacOS、.....

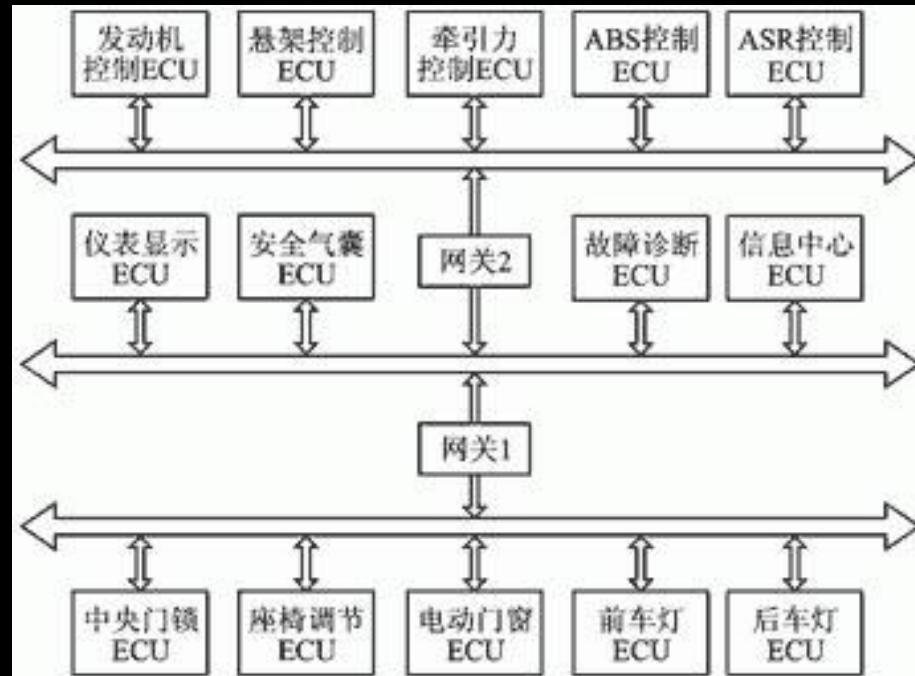


操作系统的硬件环境 —— 智能手机

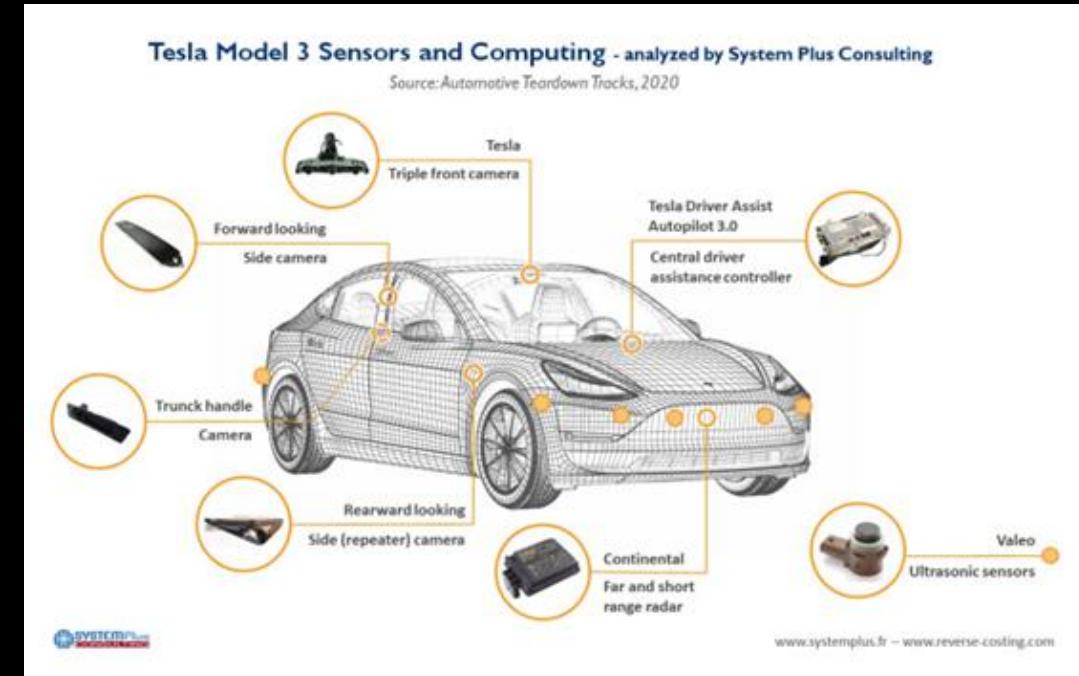


- 核心计算环境：CPU、GPU、RAM、ROM、.....
- 外设：基带芯片、屏幕、摄像头、耳机.....
- 操作系统：Android、iOS、.....

操作系统的硬件环境 —— 汽车&互联网汽车



- 核心计算环境：ECU、CAN-BUS、.....
- 外设：各种传感器.....

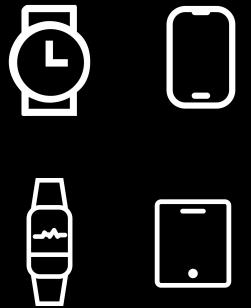


- 核心计算环境：SoC、GPU、CPU、.....
- 外设：摄像头、雷达、超声波传感器、.....



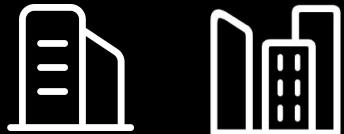
操作系统的硬件环境 —— 物联网典型场景

消费电子



穿戴设备/家居

建筑智能



智慧酒店 智慧楼宇

产业应用



车路协同 智慧农业
智能制造 智慧能源

城市及公共设施

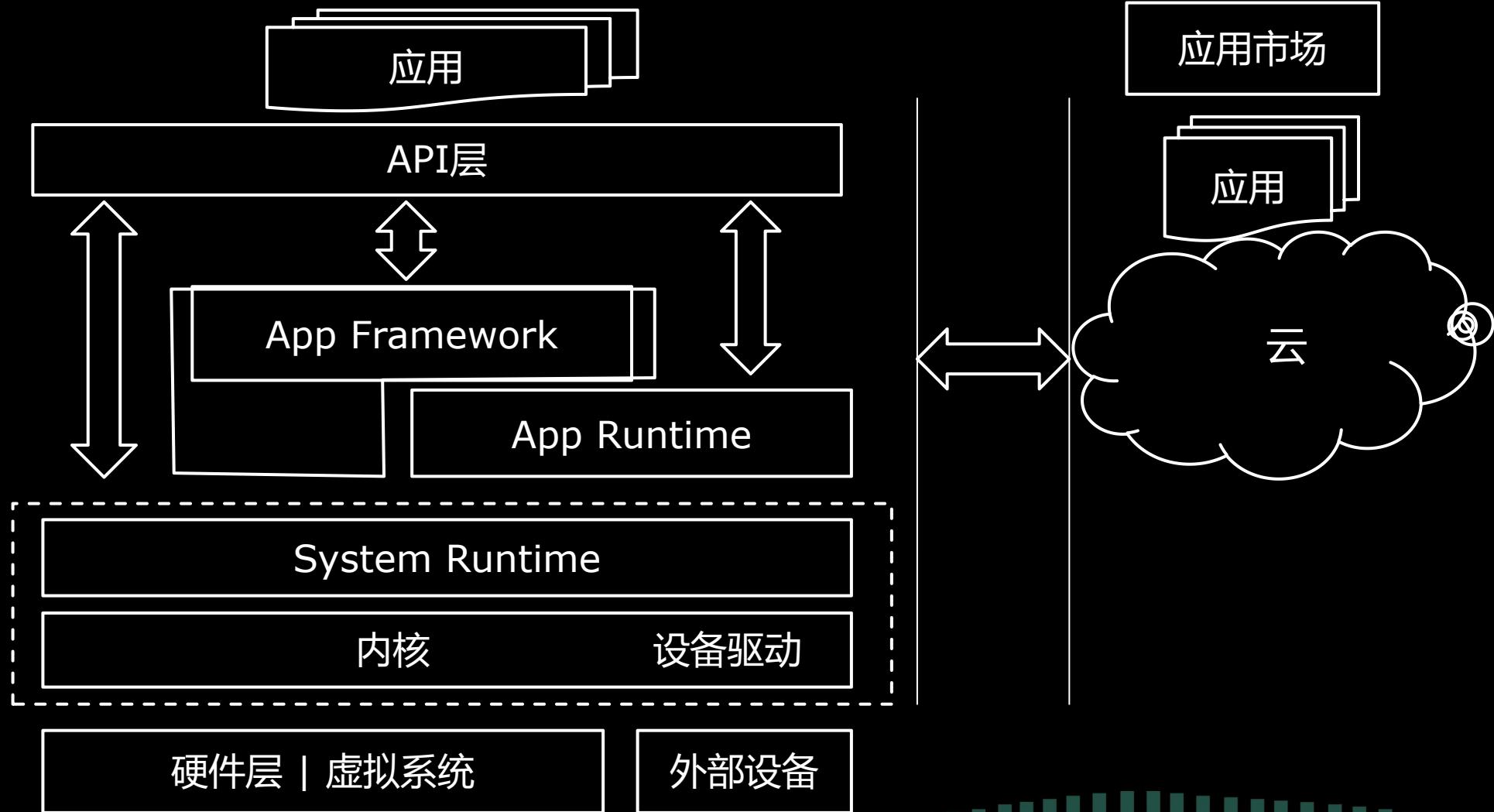


智慧城市 5G与路灯杆

- 核心计算环境：SoC、服务器（本地、边缘、云）.....
- 网络环境：以太网、各种无线网、串行网络
- 外设：各种传感器.....

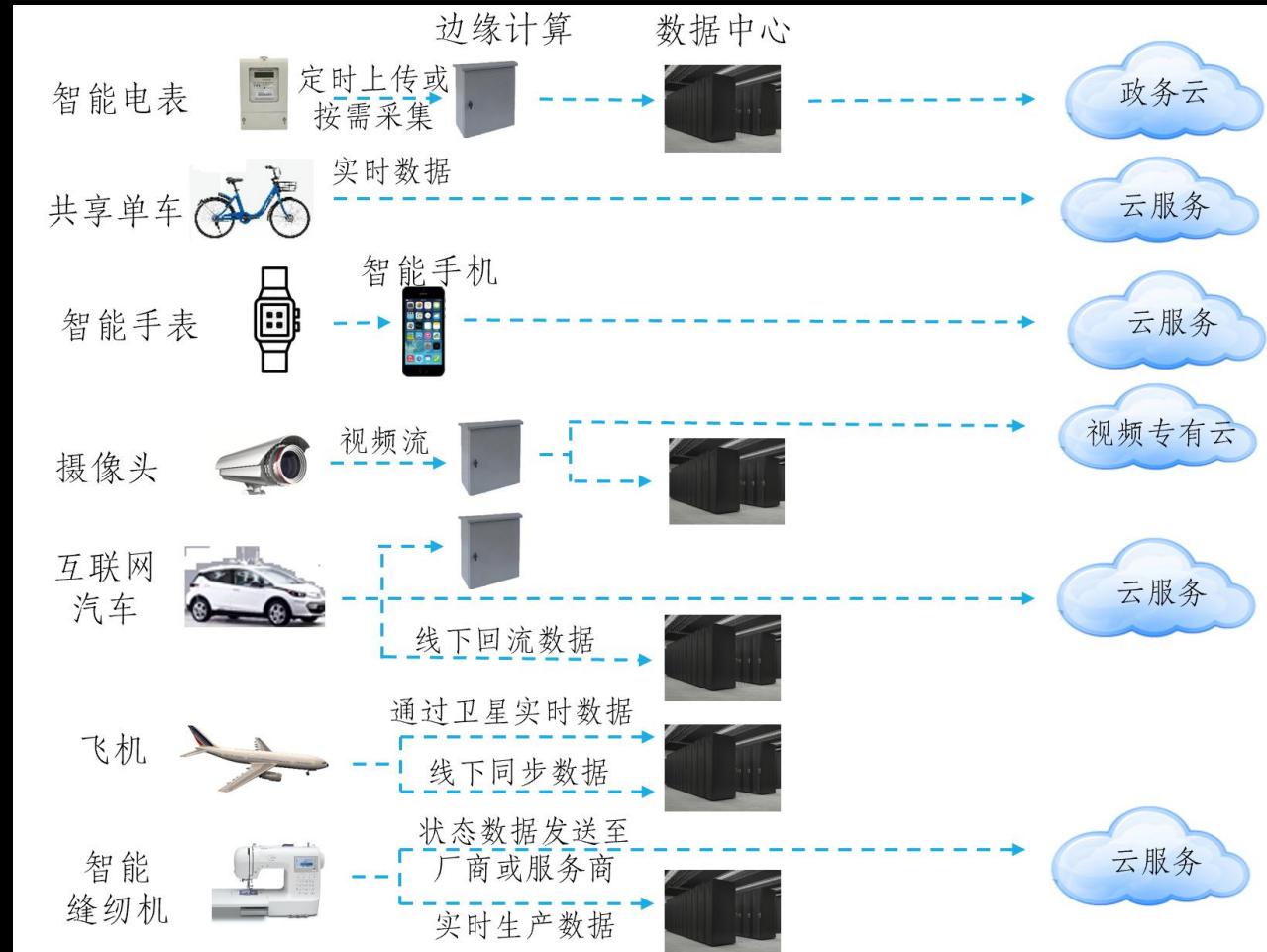
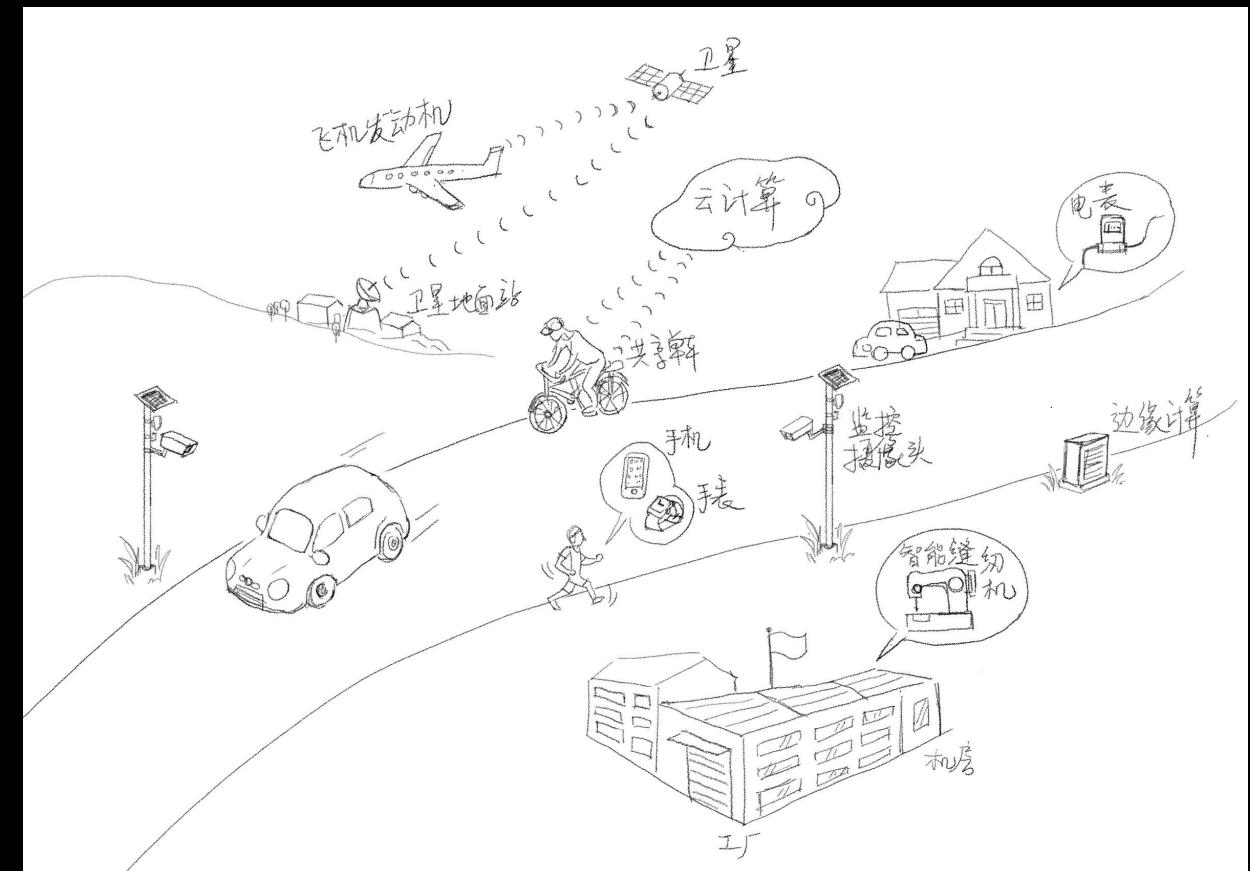


现代操作系统架构图





物联网场景的特征 —— 多样化、分散





物联网场景的特征 —— 连接

- 物联网是互联网的扩展与延伸





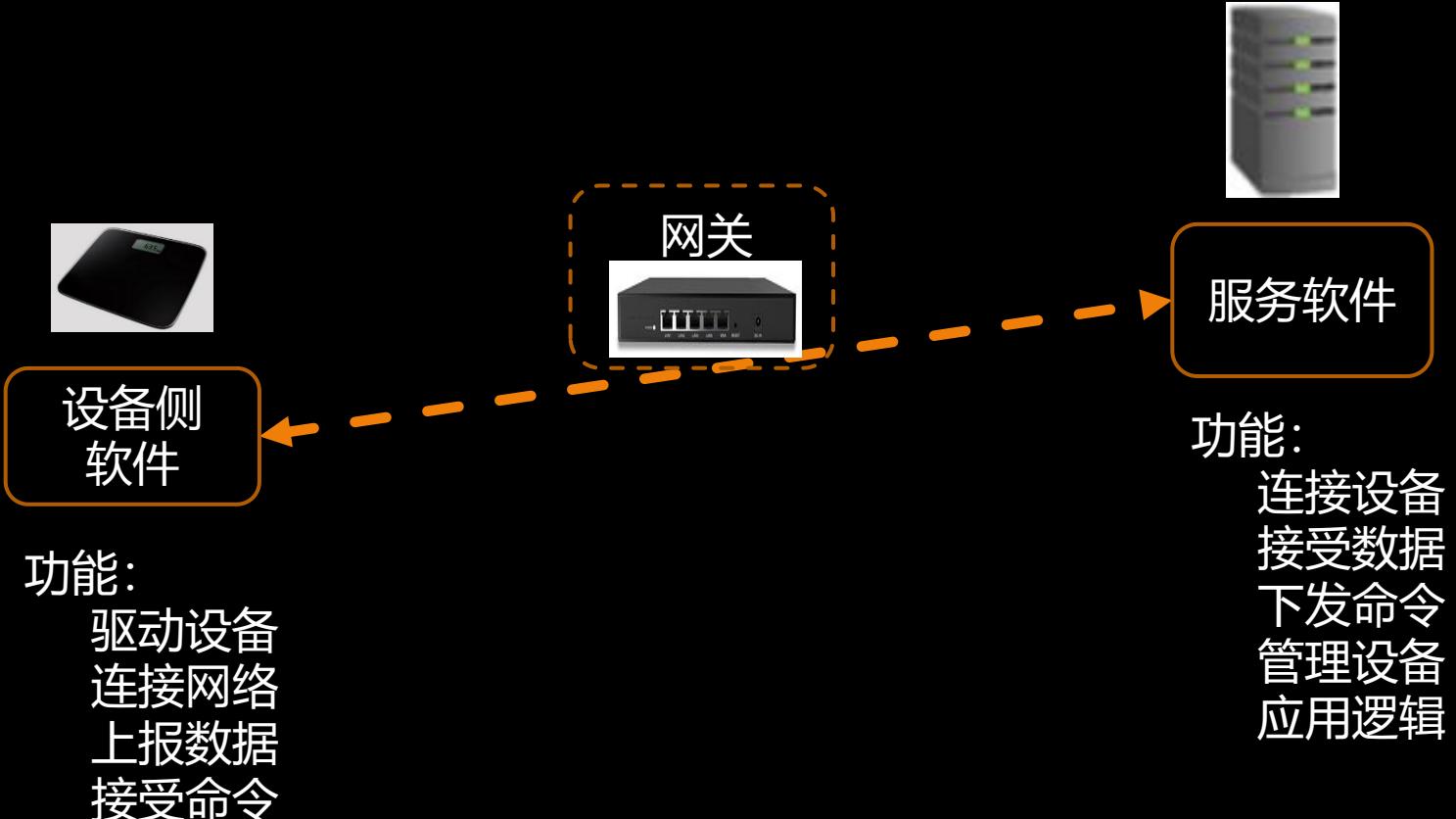
物联网场景的特征 —— 数据

- 物联网的本质是，更加深入的数字化 ==> 物理世界的数字化
- 物联网的“小数据-大数据模型”



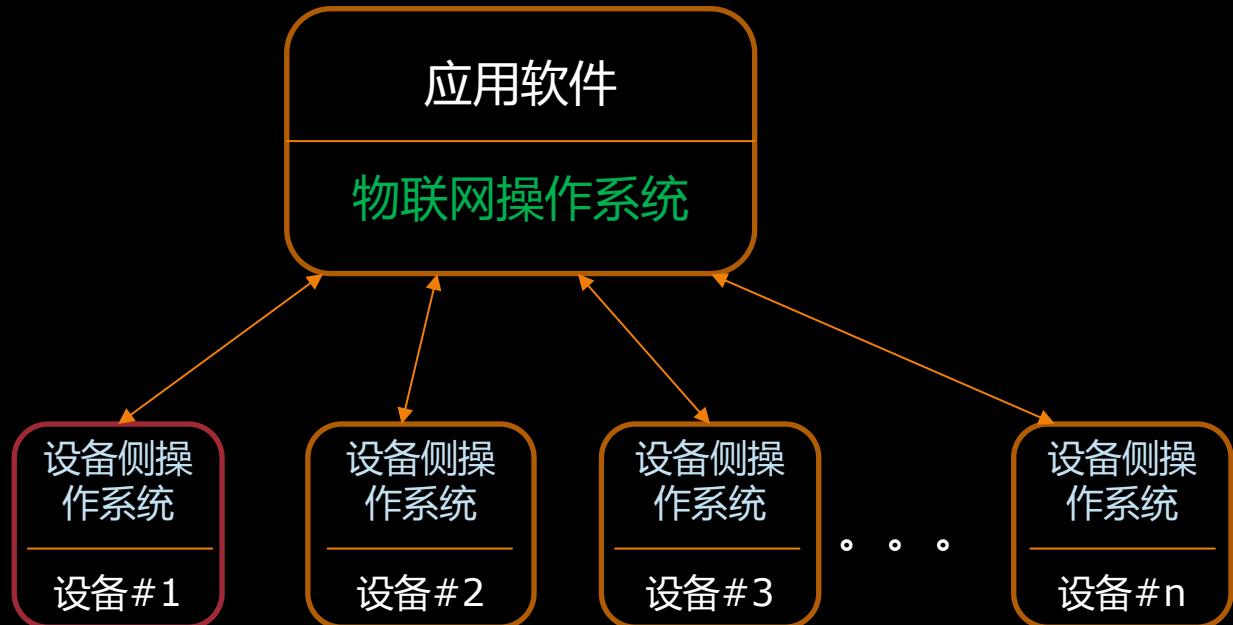


物联网场景中软件的功能抽象





物联网操作系统 —— 定义



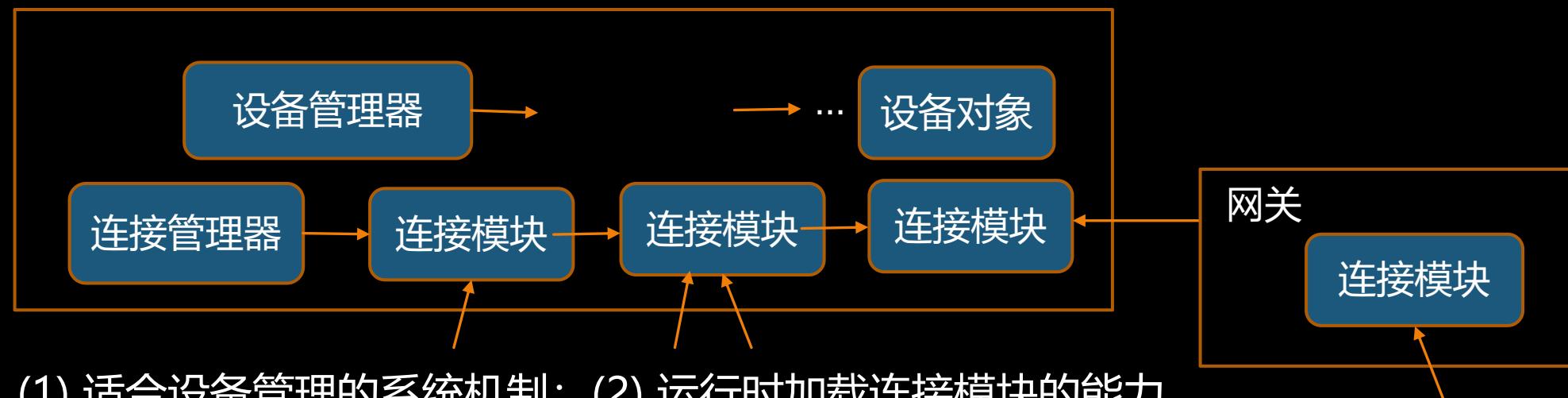
定义：物联网操作系统，是指在一个物联场景中，管理和控制该场景中各种硬件和软件资源的系统软件；也就是说，它需要管理和控制该场景中各种物联网设备和计算硬件，同时也支撑该场景中的上层应用需求。

注：设备侧操作系统可能非常简单，只是上报一个简单的状态，也可能非常复杂，本身有复杂的功能逻辑。



物联网操作系统 —— 设备连接&管理

- 术语：设备类、设备ID 、设备对象
- 设备管理器：管理设备配置信息、检索设备
- 连接管理器：负责管理设备连接。
- 设备连接位于独立的模块中，可动态配置。协议解析可内置在连接模块中。

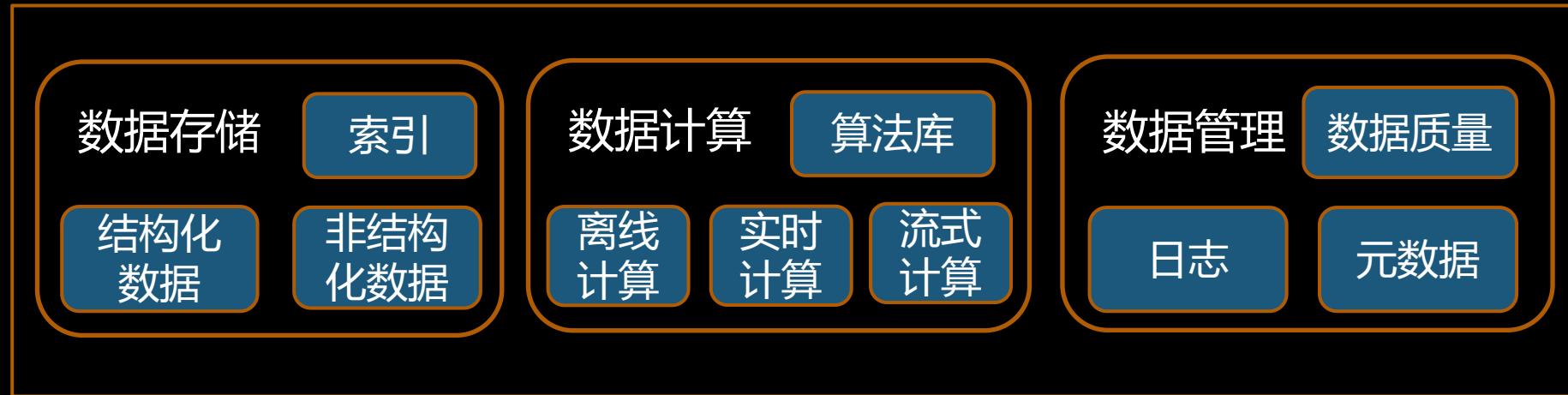


要点：(1) 适合设备管理的系统机制；(2) 运行时加载连接模块的能力



物联网操作系统 —— 数据存储&计算

- 构建一个小数据平台：基于常规的文件系统、数据库等存储技术，结合轻量化的计算引擎，将物联网设备的数据管理起来，并提供基本的计算能力

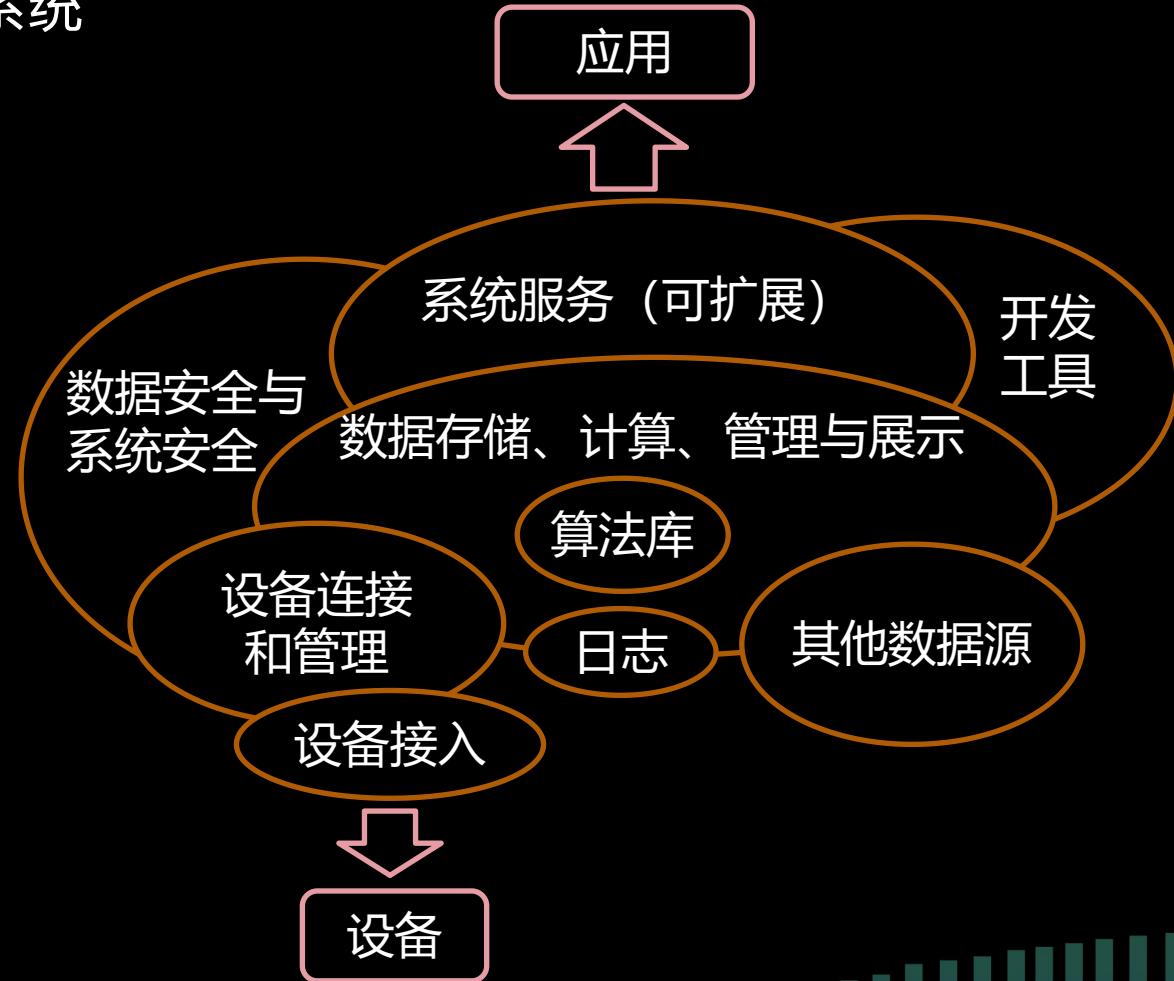


要点：(1) 物联网设备产生的数据的多样性；(2) 考虑是否采用分布式架构



物联网操作系统 —— 基础模块架构

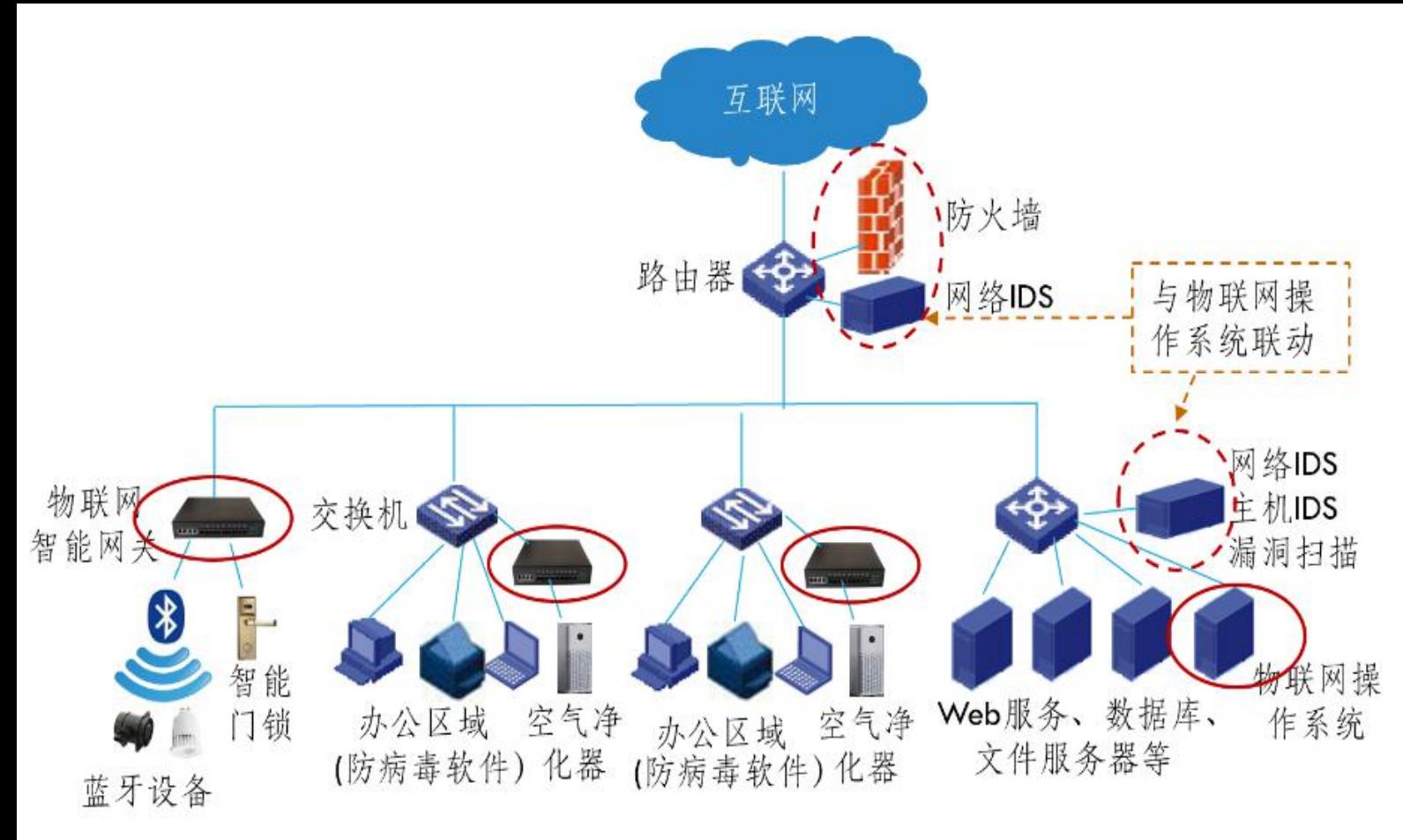
- 构建一个完整的系统





物联网操作系统 —— 物联网安全

- 设备本身的安全不可控
- 设备连接的第一跳防护；
物联网网关的安全防护
- 物联网网关、物联网操作
系统，与防火墙等设施的
联动

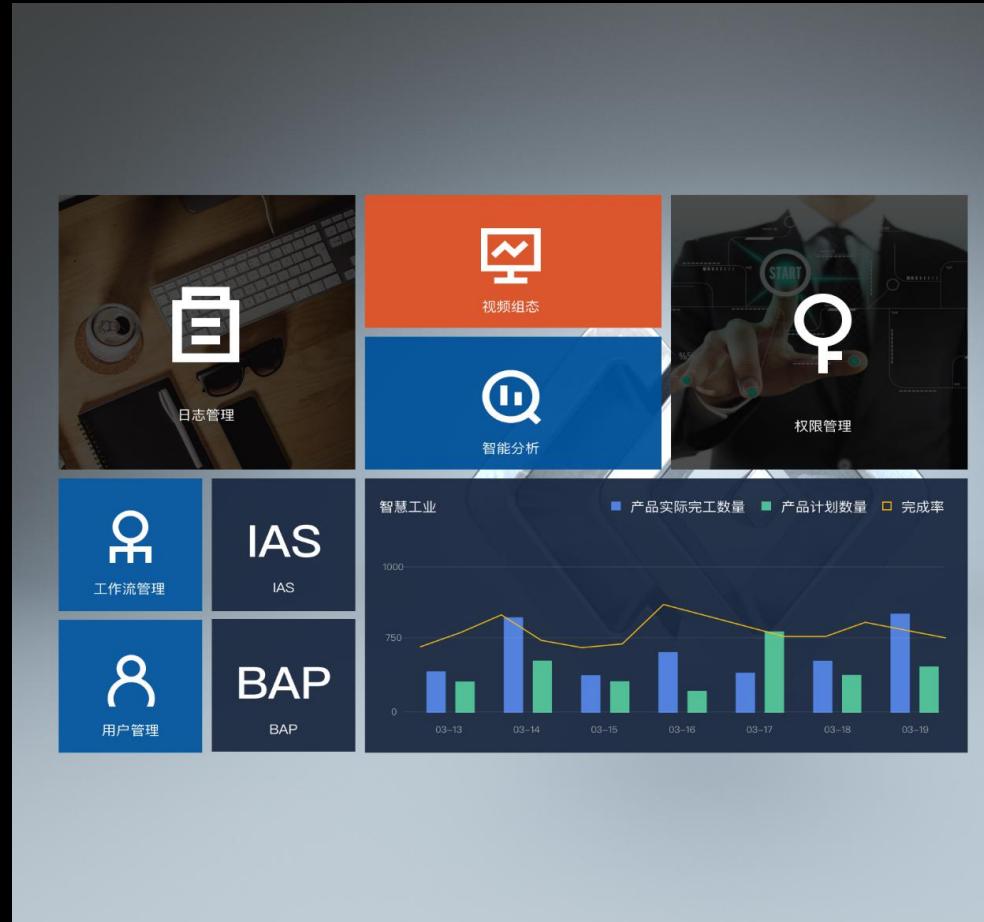


物联网安全建议方案



指令集物联网操作系统 —— iSysCore OS

- 针对各种物联场景的通用物联网操作系统
- 除了设备连接、数据能力以外，也是一个业务开发平台 —— 赋能业务
- 兼容云原生架构，支持本地部署和云部署





指令集物联网操作系统 —— iSysCore OS系统架构

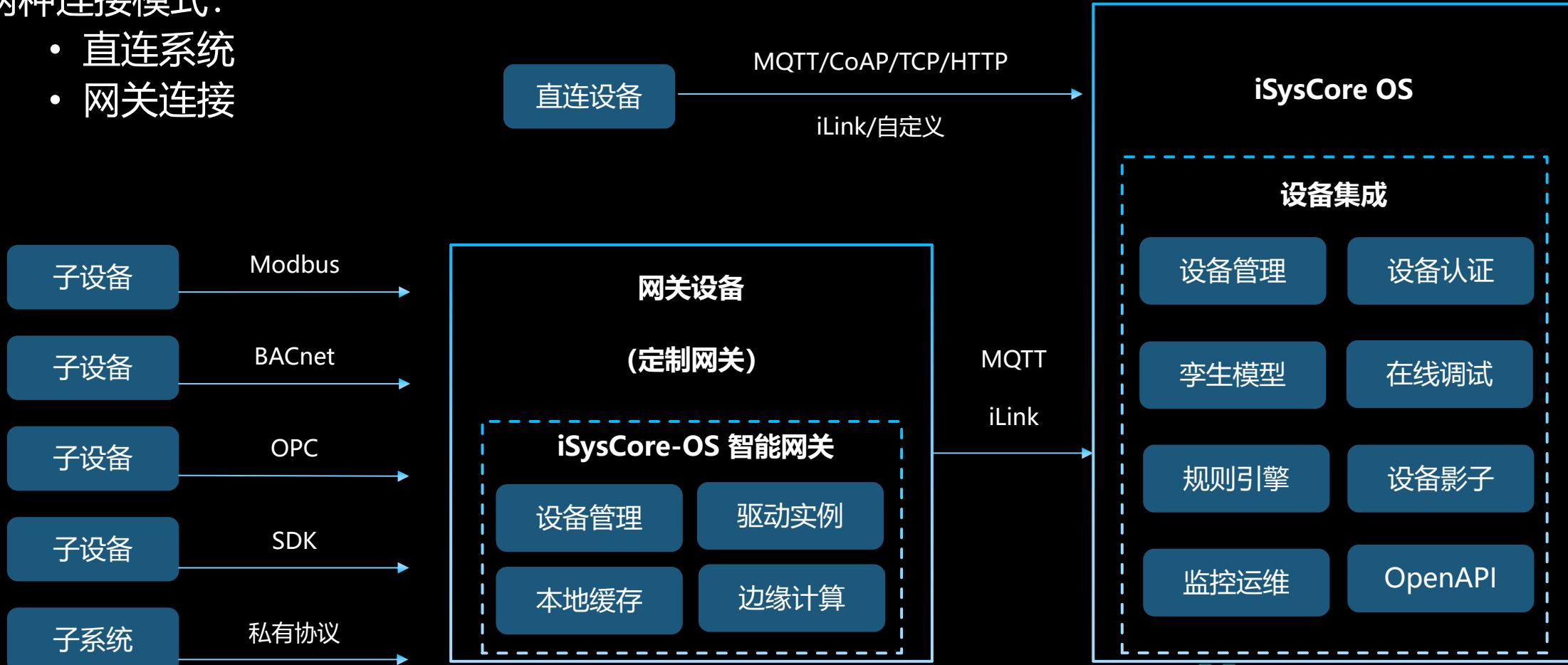




指令集物联网操作系统 —— iSysCore OS设备集成

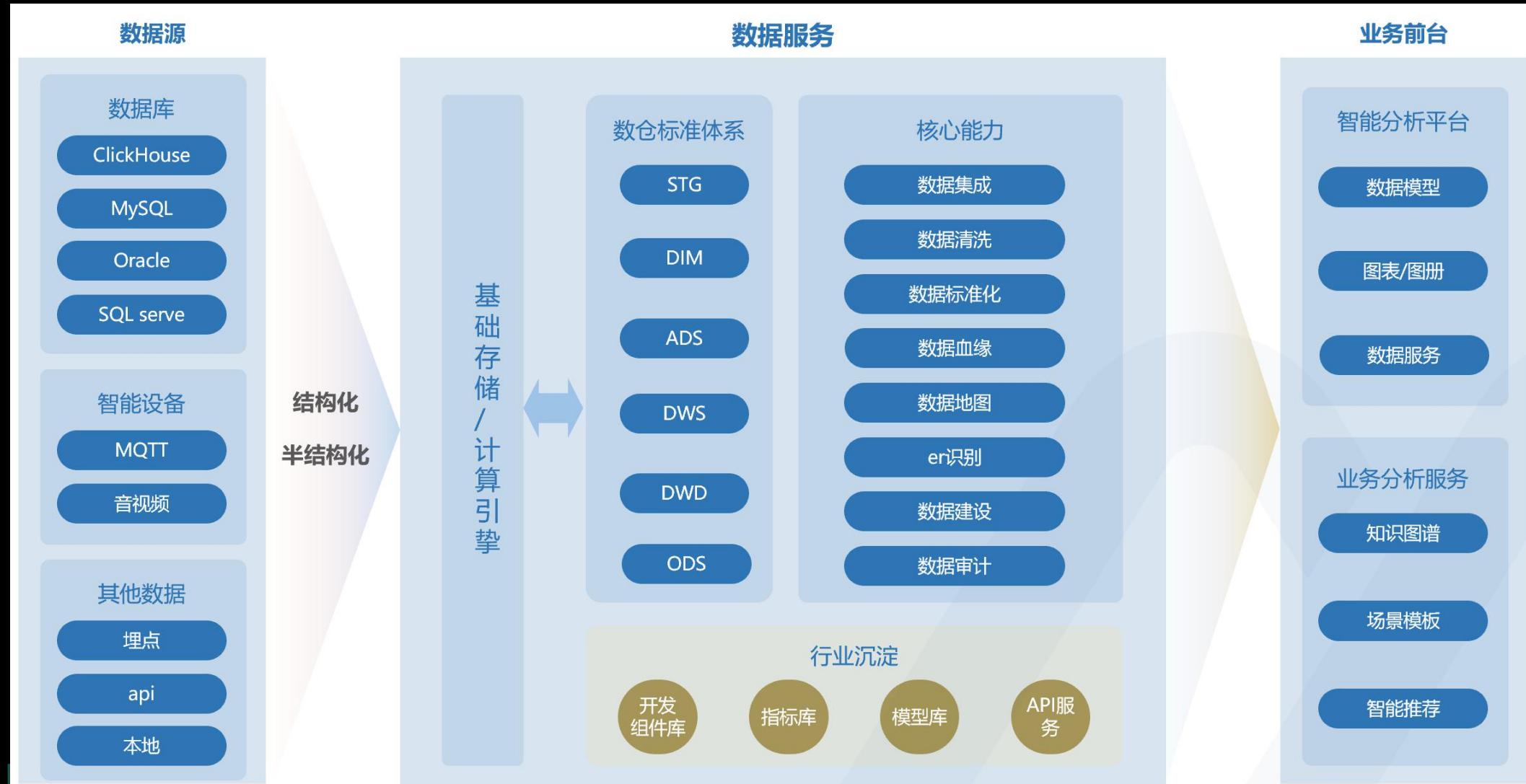
两种连接模式：

- 直连系统
- 网关连接



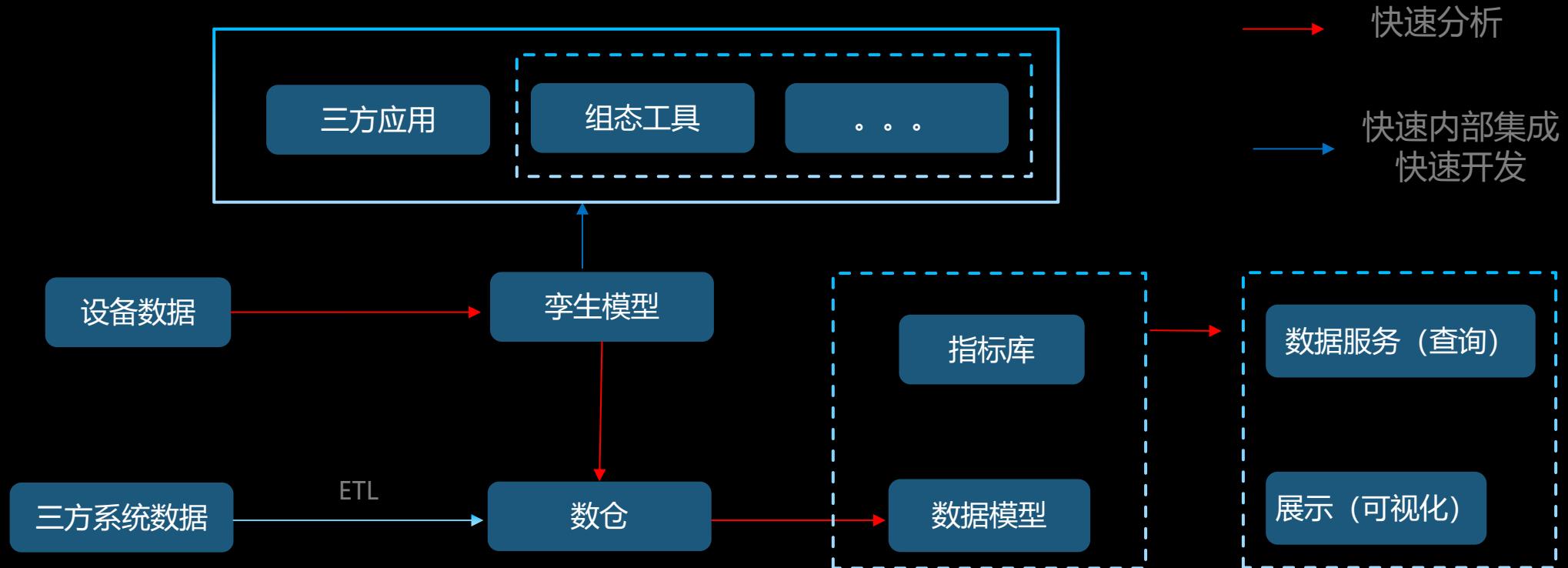


指令集物联网操作系统 —— 全生命周期数据自助管理服务



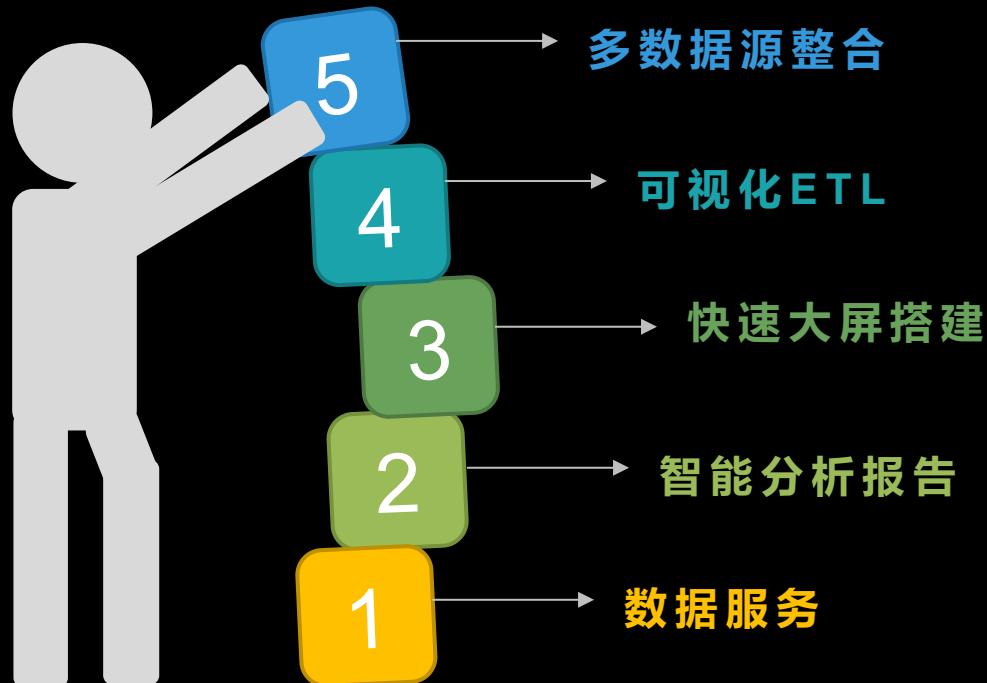


iSysCore OS —— 设备与数据打通，实现快速开发，快速集成，快速分析服务





iSysCore OS —— 满足不同客户对于数据服务的诉求



业务分析 (BAP)

产品定位：通过沉淀行业指标库，快速搭建数据大屏
目标用户：业务人员

知识图谱 智能推荐 智能报告



数据分析 (IAS)

产品定位：行业通用的数据分析与挖掘自助服务平台。
目标用户：数据分析师

设备数据的直接接入 数据源接入 数据建模 数据可视化



数据管理 (UDMP)

产品定位：数据集成、治理与开发平台，帮助企业搭建中台，为各个场景提供数据服务。
目标用户：数据开发人员

指标库 主数据管理 数据工厂 数据质量 数据标准



iSysCore OS —— 服务中枢(PIVOT)

- 基于微服务、容器、K8S等云原生技术，可实现快速部署，动态扩展，稳定运行
- 提高故障排查效率，提升运营与维护的便捷性



快速部署

开发流水线

全链路监控



iSysCore OS —— 开发能力/SDK

- 以OpenAPI或集成SDK的方式进行开发；
- 单点登录/权限/组织架构/公共服务；
- 设备管理/设备查询/设备控制

- IAS：提供数据建模及可视化的开发能力，可以灵活的搭建BI报表/数据大屏；并可以以API的方式提供数据服务；
- BAP/指标库：通过公式配置的方式编制数据指标，并提供对应的数据服务和可视化展示；
- UDMP：提供数据ETL和数据管理的能力

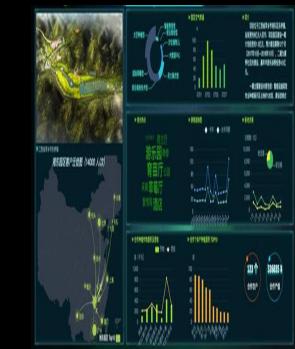
- 动态表单：可视化配置表单，可以被集成到业务系统中，快速完成中后台管理平台类的业务应用，并且满足业务快速定制迭代的需求；
- 工作流：可视化编制业务工作流，结合动态表单快速的完成业务流程的实现和定制开发；

- Pivot：对基于微服务开发的应用提供运维、监控、限流和扩容的能力





指令集物联网操作系统 —— 构建行业操作系统



智慧楼宇

智慧建筑

智能制造

智慧农业

智慧矿山

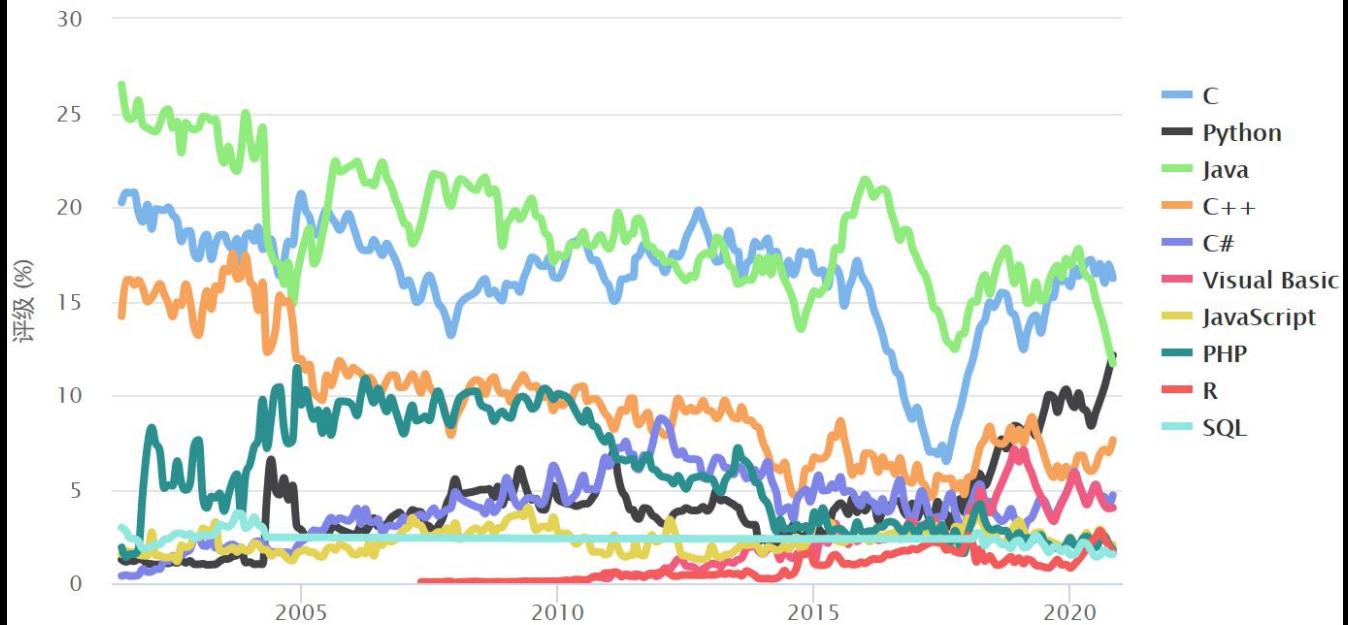
智慧景区

面向行业或领域的智能操作系统

指令集物联网操作系统

TOP 10 编程语言的走势图

Source: www.tiobe.com

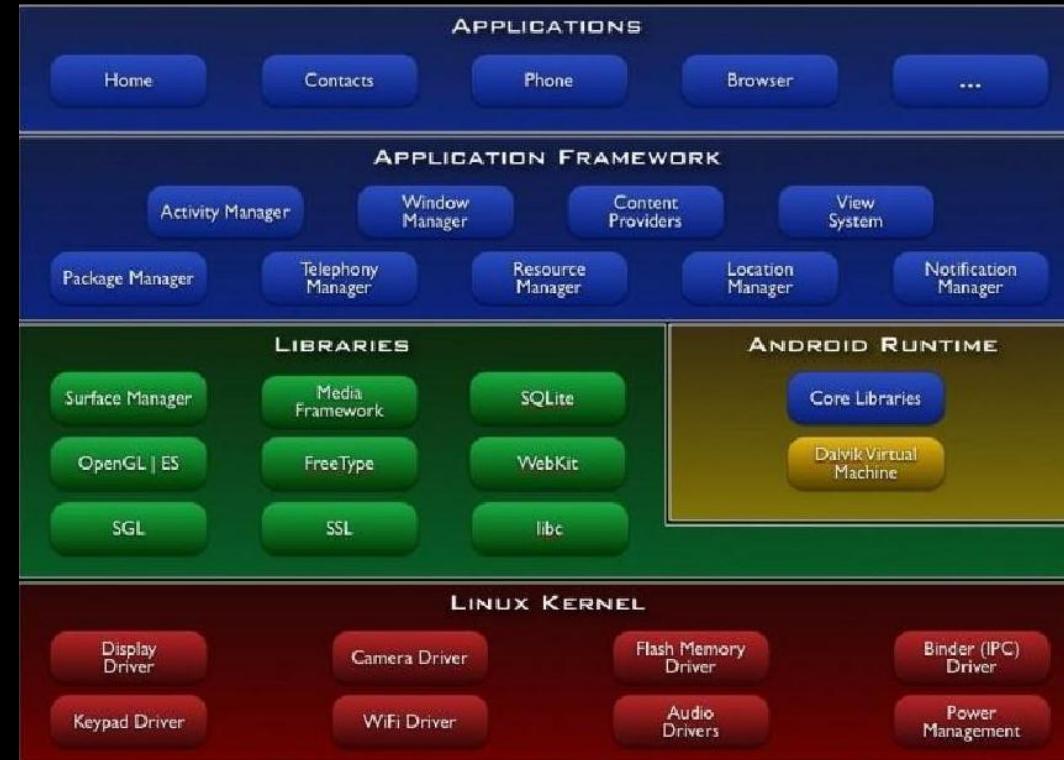


编程语言排行榜 TOP 50 榜单

| 排名 | 编程语言 | 流行度 | 对比上月 | 年度明星语言 |
|----|--------------|--------|---------|------------------|
| 1 | C | 16.21% | ▼ 0.74% | 2017, 2008, 2019 |
| 2 | Python | 12.12% | ▲ 0.84% | 2010, 2007, 2018 |
| 3 | Java | 11.68% | ▼ 0.88% | 2015, 2005 |
| 4 | C++ | 7.60% | ▲ 0.66% | 2003 |
| 5 | C# | 4.67% | ▲ 0.51% | |
| 6 | Visual Basic | 4.01% | ▲ 0.04% | |
| 7 | JavaScript | 2.03% | ▼ 0.11% | 2014 |



操作系统中代码的语言分布 —— Android



C++ < C < Java



操作系统中代码的语言分布 —— iSysCore OS

整体代码包：

C/C++: 20%
Go: 10%
Java: 40%
JS/CSS: 25%
其他: 5%

自研部分：

Java: 60%
JS/CSS: 30%
Python: 3%
Go: 1%
C: 1%
其他: 5%

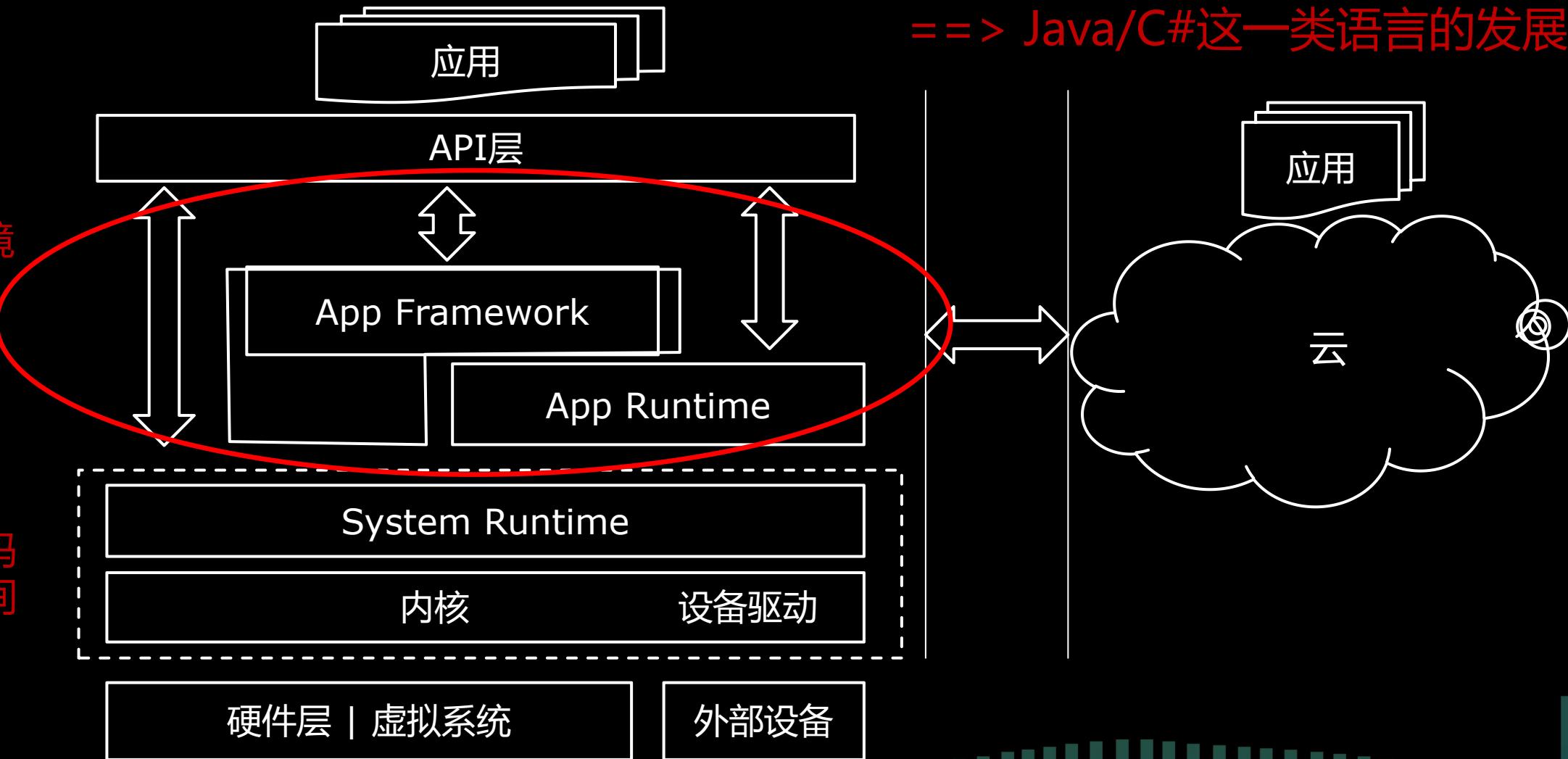
C/C++ < JS/CSS < Java



操作系统 vs. 语言的发展趋势一

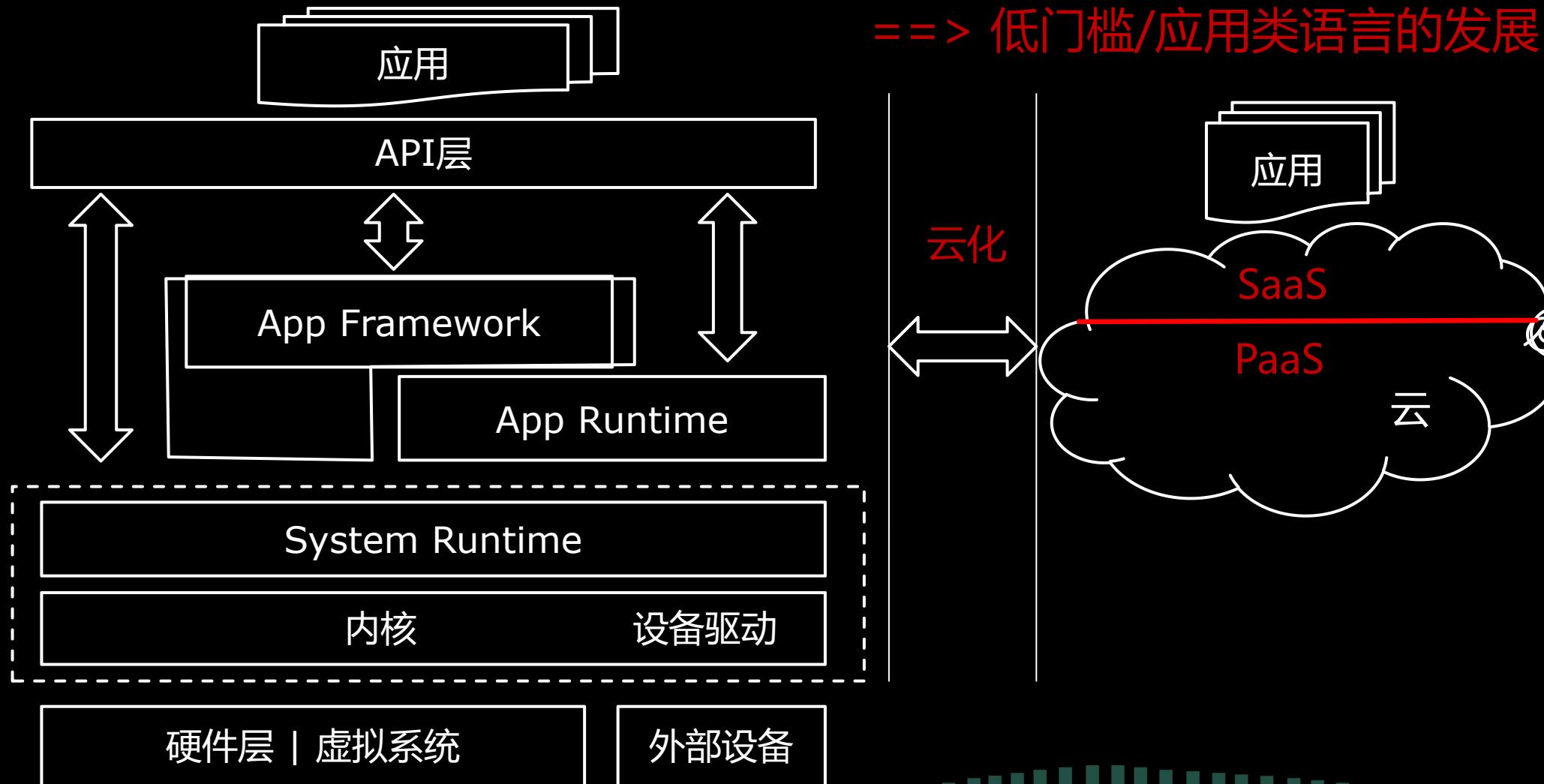
字节码
虚拟机环境

二进制代码
机器码空间



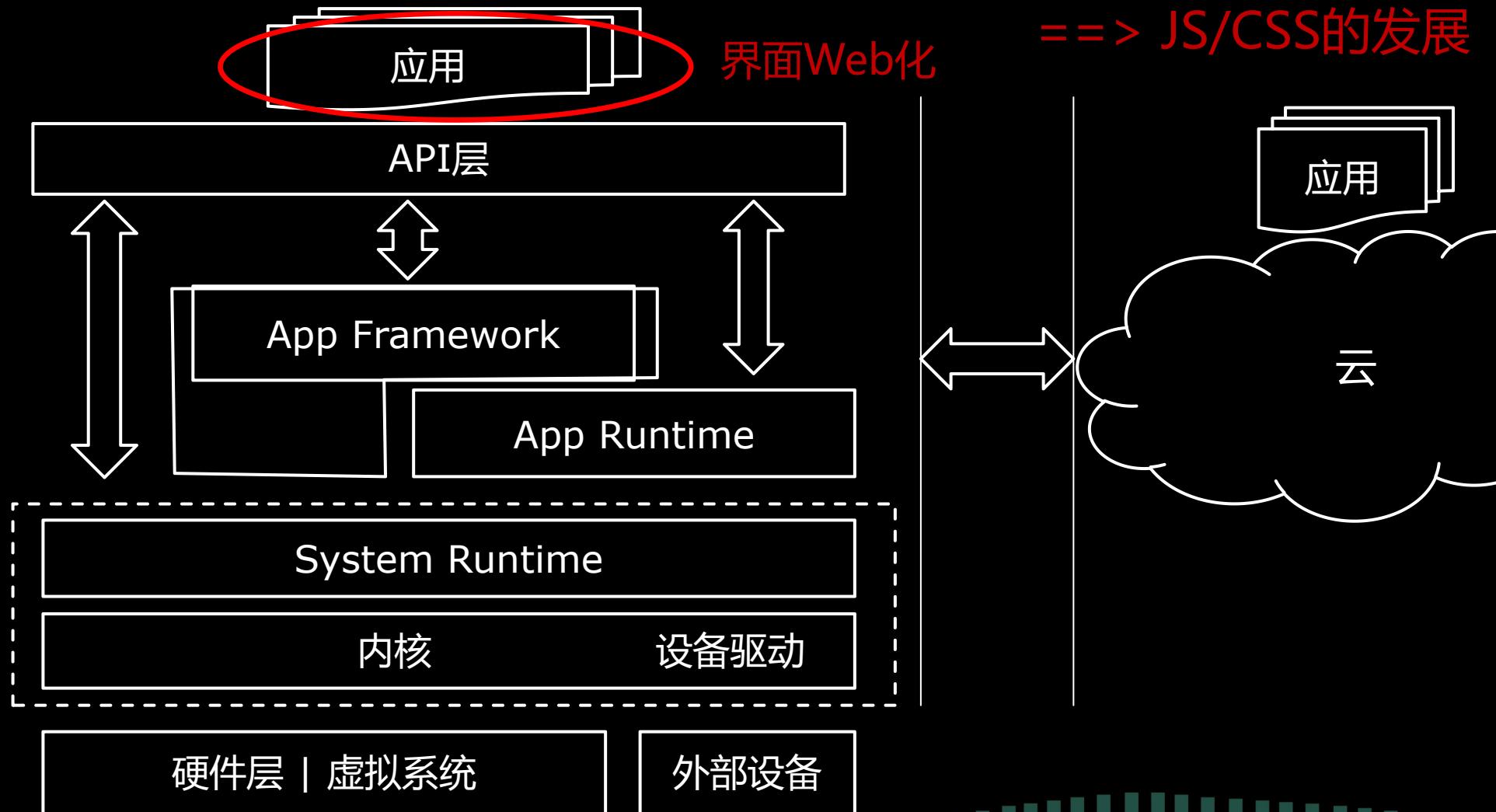


操作系统 vs. 语言的发展趋势二



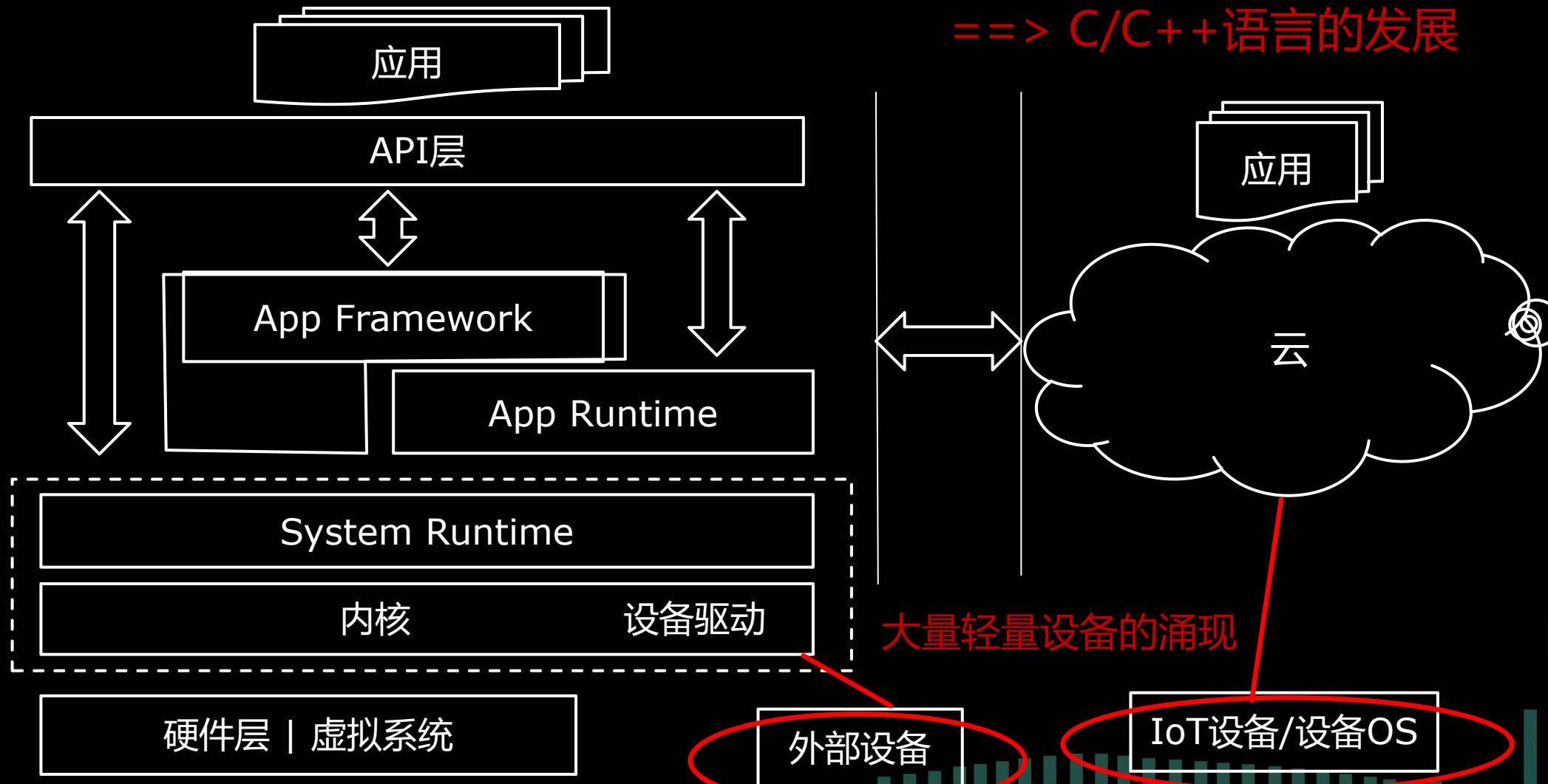


操作系统 vs. 语言的发展趋势三





操作系统 vs. 语言的发展趋势四





操作系统 vs. 语言的发展趋势的关系

Runtime环境趋势

云化

界面Web化

轻量设备涌现

| 编程语言排行榜 TOP 50 榜单 | | | | |
|-------------------|--------------|--------|---------|------------------|
| 排名 | 编程语言 | 流行度 | 对比上月 | 年度明星语言 |
| 1 | C | 16.21% | ▼ 0.74% | 2017, 2008, 2019 |
| 2 | Python | 12.12% | ▲ 0.84% | 2010, 2007, 2018 |
| 3 | Java | 11.68% | ▼ 0.88% | 2015, 2005 |
| 4 | C++ | 7.60% | ▲ 0.66% | 2003 |
| 5 | C# | 4.67% | ▲ 0.51% | |
| 6 | Visual Basic | 4.01% | ▲ 0.04% | |
| 7 | JavaScript | 2.03% | ▼ 0.11% | 2014 |



- 一门完美的语言
- 融合了多种设计范式
- 集大成的完美语言并不是最实用的
- 作为最底层语言，不如C语言简单；
作为应用层语言，又不如Java/Python/VB这类语言的生产效率高
- 最适合作为中间件语言，特别是对性能有要求的中间件

C++是最好的语言！因为。。



谢谢！

蒲俊峰

腾讯广告推荐系统负责人



16年C++工程实践经验，负责腾讯广告推荐系统研发工作。

在实战中，积累了深厚的高吞吐、低时延分布式在线推理系统设计、开发经验，精通系统性能调优。

主办方：

Boolan
高端IT咨询与教育平台

CPP-Summit 2020

蒲俊峰

腾讯高级工程师

实时推荐系统 设计与优化

自我介绍

蒲俊峰

腾讯 - TEG数据平台部 - 精准推荐中心

2011年硕士毕业于电子科技大学，同年加入华为

2014年加入腾讯，从事精准推荐系统领域研发工作



议程

1 广告业务概览

业务简介及面临的挑战

2 软硬结合、追求性能的极致

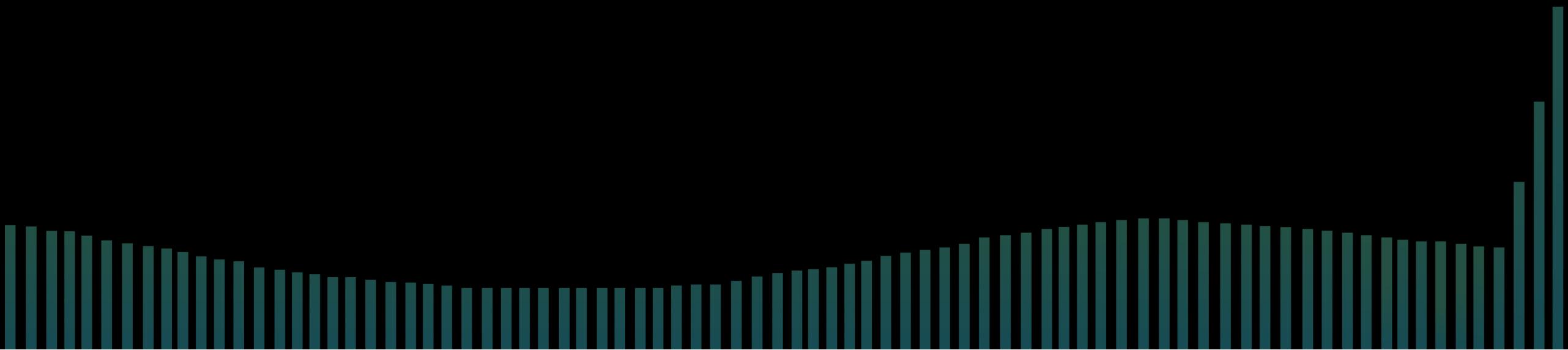
两条腿走路：量化，立体化评估机制；优化，软硬结合，架构优化

3 智能巡查看调优，保障服务质量

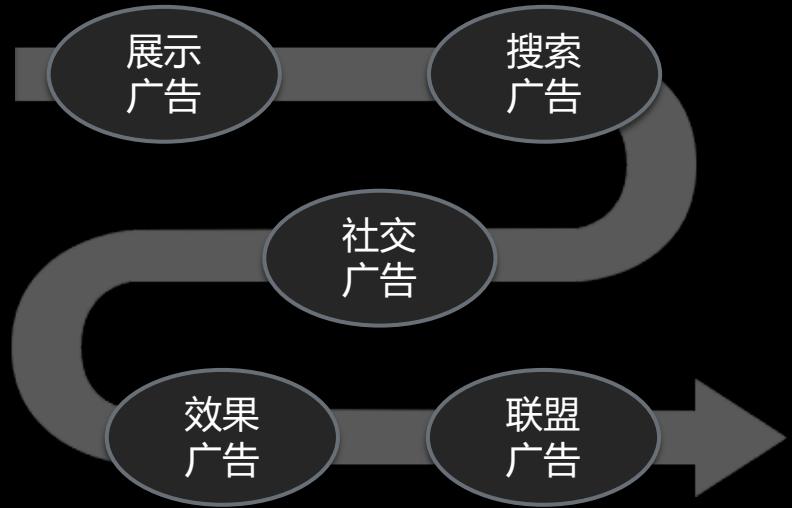
工程实践，引入AI加持

01

广告业务概览



业务简介

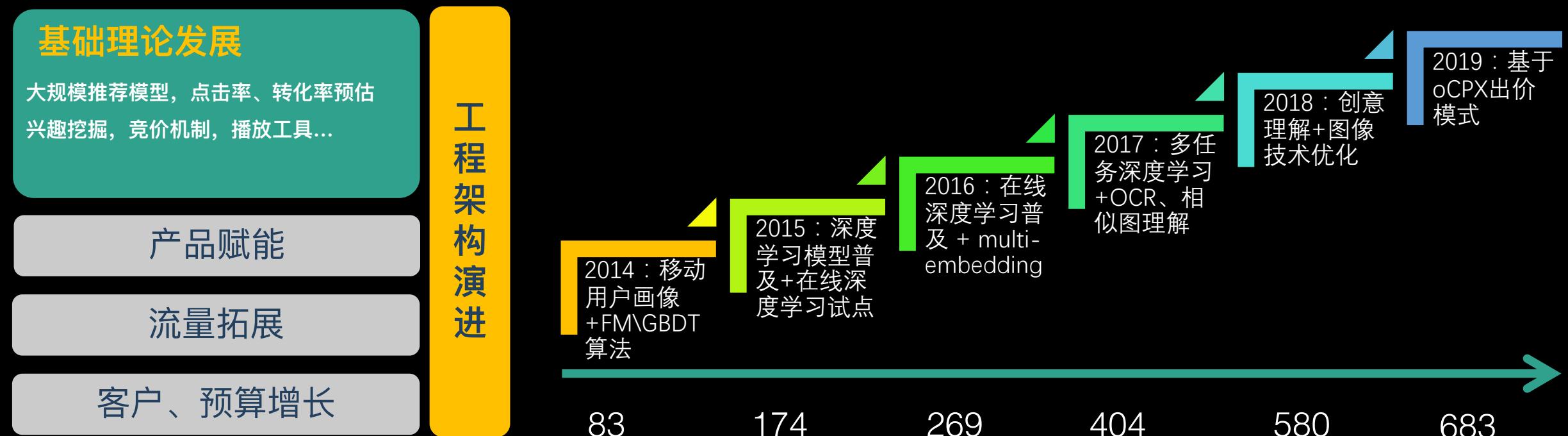


机遇&挑战

丰富的流量生态，有待持续挖掘商业价值
多样广告形态发展，推进技术架构不断演进



推荐预估的发展历程



工程&基础理论的融合，提供了广告业务发展的源动力。

深耕推荐系统，沉淀的技术栈



精准推荐技术栈

面临的挑战

大规模

- 千亿级日请求量
- 十万亿数据交互
- 万亿特征空间

低时延

- 毫秒级模型推理

高可用

- 7 * 24无间断服务
- 分级容灾机制

02

软硬结合、追求性能的极致

高性能的解决之道

特征工程

- 用户特征KV查询：长尾延时、网络延时
- 广告特征解析：嵌套PB反序列化，CPU、时耗开销压力大
- 高性能列存储：优化用户特征查询长尾

训练工程

- 突破单机模型大小限制，支持分布式高维稀疏模型训练
- 高性能稀疏参数服务，压制分布式Embedding查询长尾延时
- 异构存储：持久化内存，提升单机“内存”容量

推理工程

- Shared Nothing计算，任务分级隔离调度
- 异构计算加速：FPGA、GPU
- 全流程Pipeline加速

实验工程

- 效果影响：推荐效果直接决定广告收入的波动
- 快速实验：百级数量模型，15分钟粒度迭代更新
- 服务质量：资源隔离&保护，集群状态自动管控

量化

立体化评估机制

优化

软、硬结合

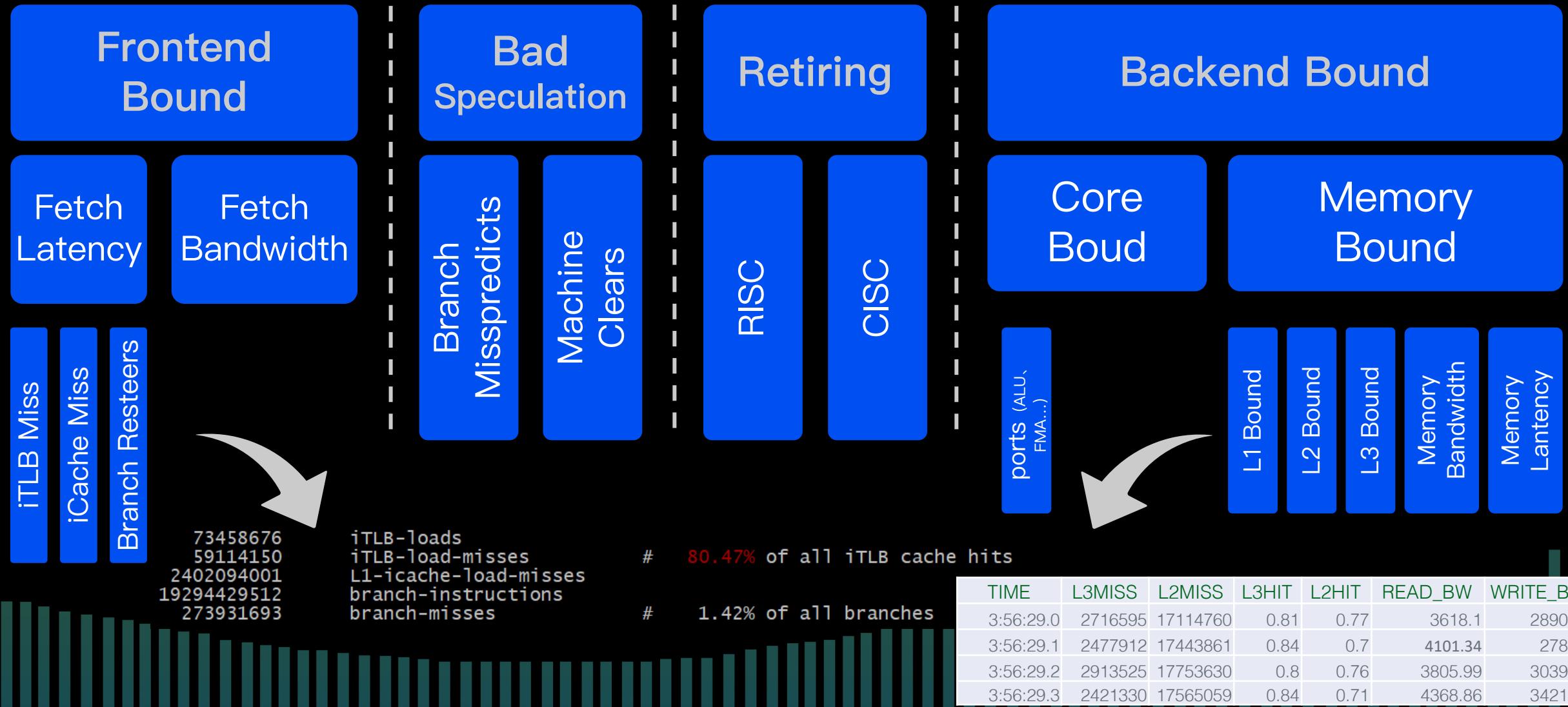
架构优化

立体化评测 – 穿透层次量化

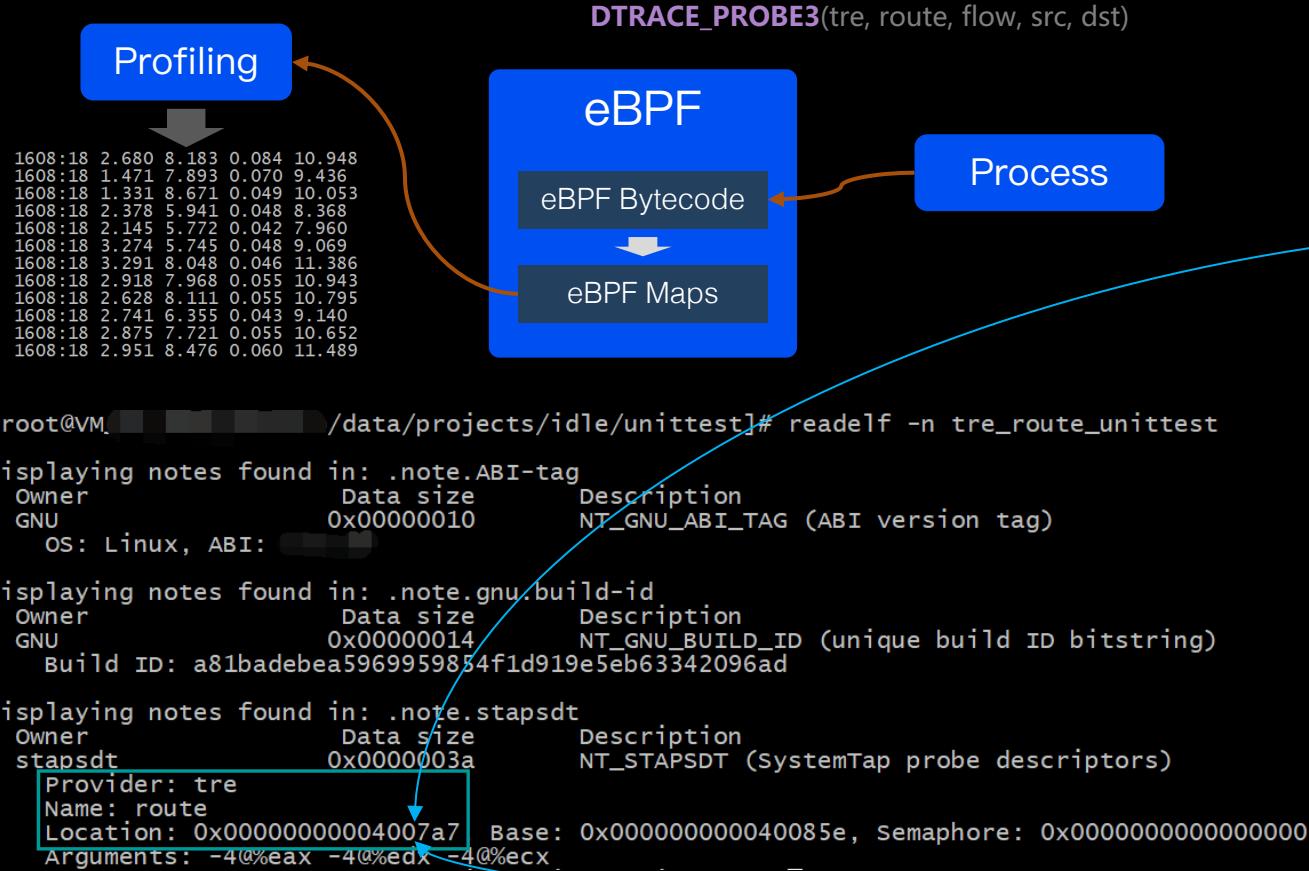


采用合理的架构，充分发挥硬件能力

立体化评测 – TMAM性能分析方法



立体化评测 – eBPF & USDT 数据采集



```
[root@VM] /data/projects/idle/unittest]# readelf -n tre_route_unittest
Displaying notes found in: .note.ABI-tag
  Owner           Data size      Description
    GNU            0x00000010  NT_GNU_ABI_TAG (ABI version tag)
OS: Linux, ABI: [REDACTED]

Displaying notes found in: .note.gnu.build-id
  Owner           Data size      Description
    GNU            0x00000014  NT_GNU_BUILD_ID (unique build ID bitstring)
Build ID: a81badebea59699598[REDACTED]4f1d919e5eb63342096ad

Displaying notes found in: .note.stapsdt
  Owner           Data size      Description
  stapsdt        0x0000003a  NT_STAPSDT (SystemTap probe descriptors)
Provider: tre
Name: route
Location: 0x000000000004007a7
Arguments: -4@eax -4@edx -4@ecx
```

```
(gdb) disassemble TRERoute::route_dag
Dump of assembler code for function TRERoute::route_dag(DAGInterface):
0x00000000000400792 <+0>: push %rbp
0x00000000000400793 <+1>: mov %rsp,%rbp
0x00000000000400796 <+4>: sub $0x20,%rsp
0x0000000000040079a <+8>: mov %rdi,-0x18(%rbp)
0x0000000000040079e <+12>: mov -0x4(%rbp),%eax
0x000000000004007a1 <+15>: mov -0x8(%rbp),%edx
0x000000000004007a4 <+18>: mov -0xc(%rbp),%ecx
0x000000000004007a7 <+21>: int3
0x000000000004007a8 <+22>: mov $0x400850,%edi
0x000000000004007ad <+27>: callq 0x400680 <puts@plt>
0x000000000004007b2 <+32>: mov $0x0,%eax
0x000000000004007b7 <+37>: leaveq
0x000000000004007b8 <+38>: retq
End of assembler dump.
```

在线推理的挑战

推理计算三要素

1. 数据（特征）：

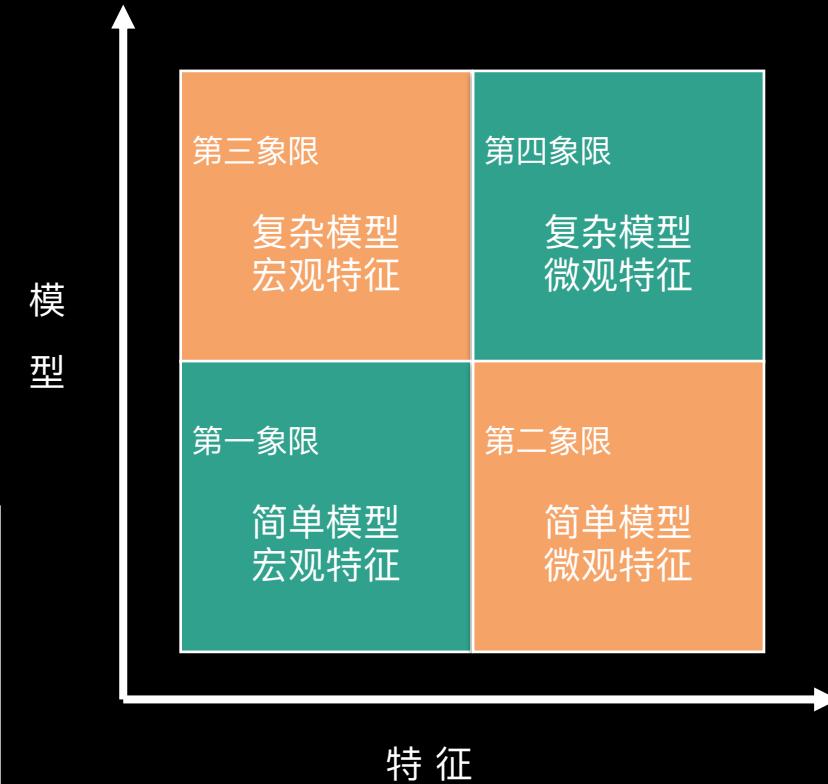
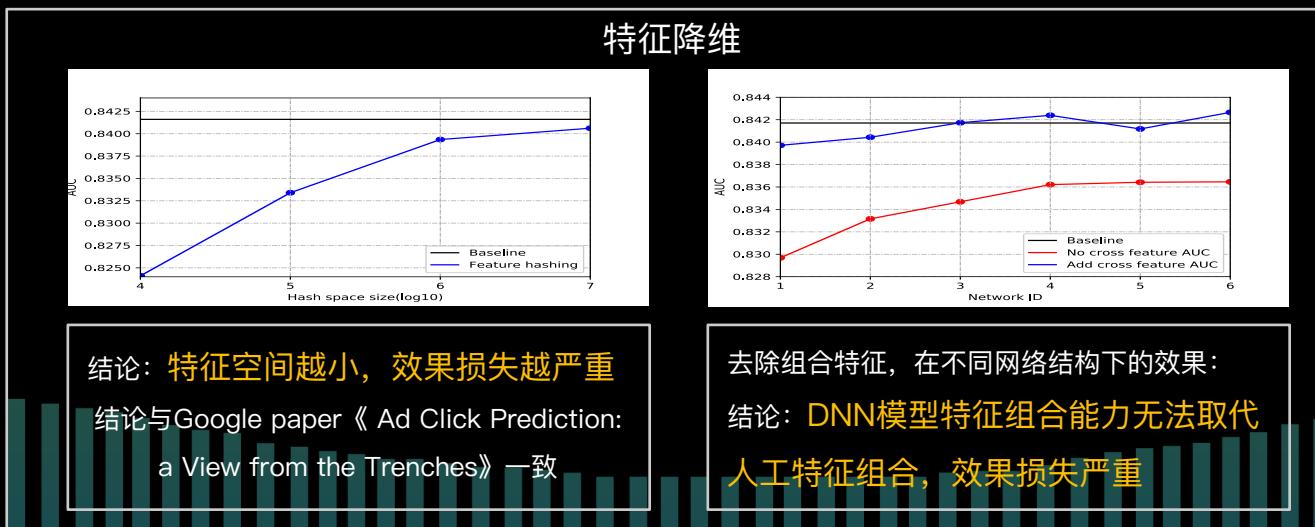
计算数据（队列）越大，预测效果越精准
大规模离散特征、组合特征

2. 模型：

模型结构复杂化；模型空间超大化

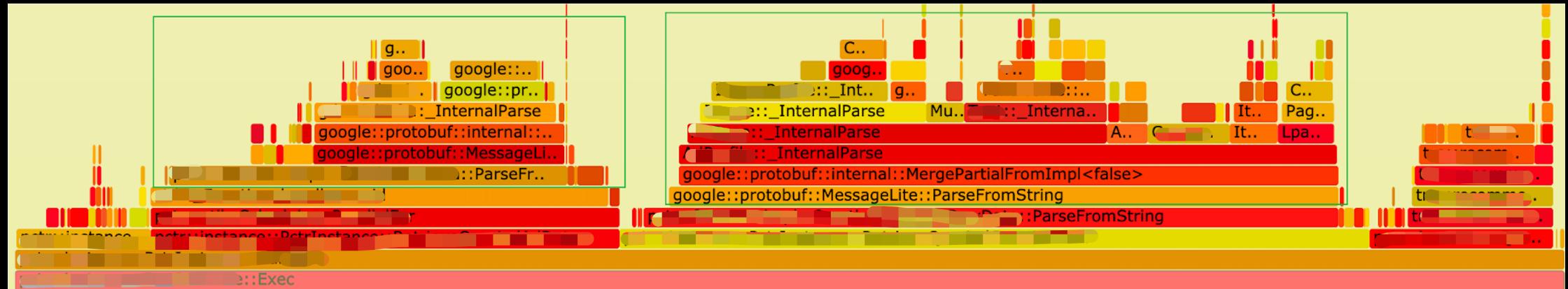
3. 算力：

计算量增加带来更大算力需求



数据 – 广告特征解析瓶颈

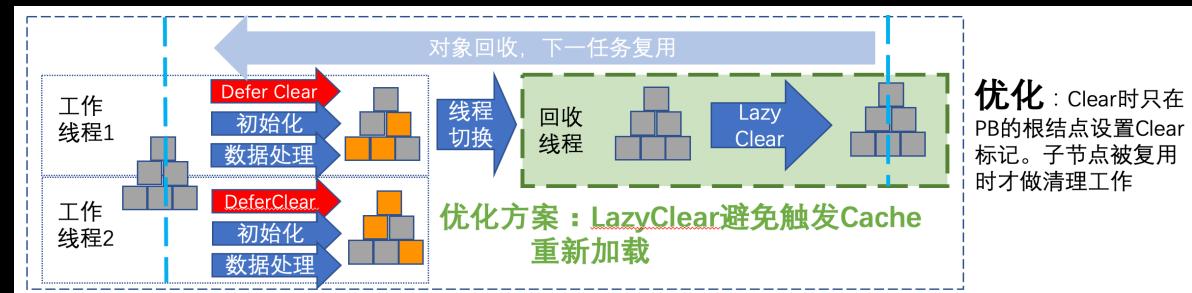
- 特征抽取是计算引擎必备的“热身”活动
- 特征PB对象的拼接、反序列化开销逐步成为系统瓶颈



- 性能瓶颈原因：
 - 大量PB对象申请、释放（及clear操作）在多线程间漂移
 - 多层嵌套repeated message，解析时add消息对象开销
 - PB反序列化

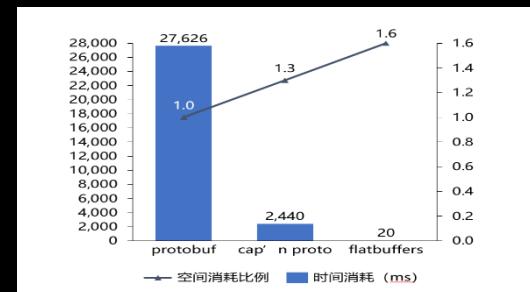
数据 - 离线解析、实时计算

- PB特性加速解析
 - PB LazyClear编译优化，CPU缓存优化，Arena缓存
 - PB对象定义打平
 - 关闭PB Unknown对象解析



- 快速协议
 - FlatBuffer、Cap'n proto等

特征KV存储成本1.6倍

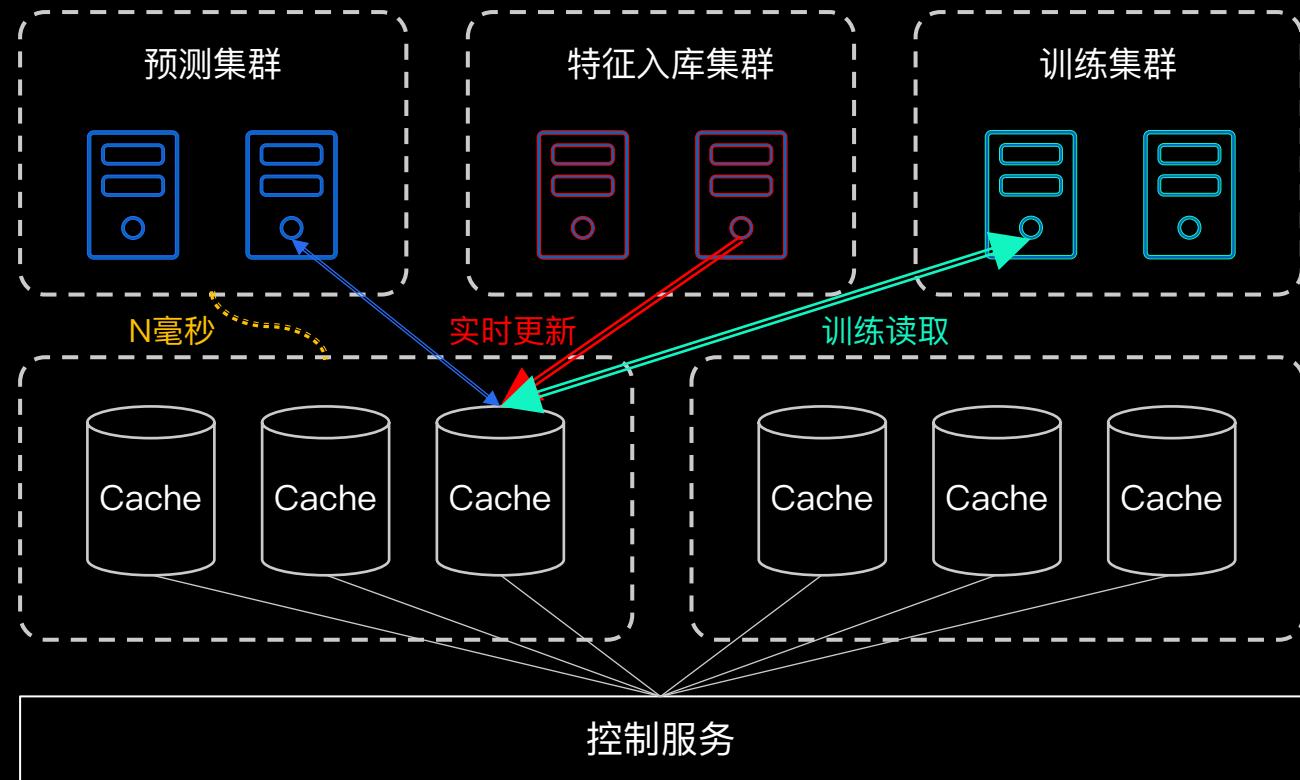


- 离线化解析
 - 自定义二进制协议，推荐计算直接使用协议对象
 - 广告特征主动推送、主动解析
 - 基于共享内存，解决冷启动问题



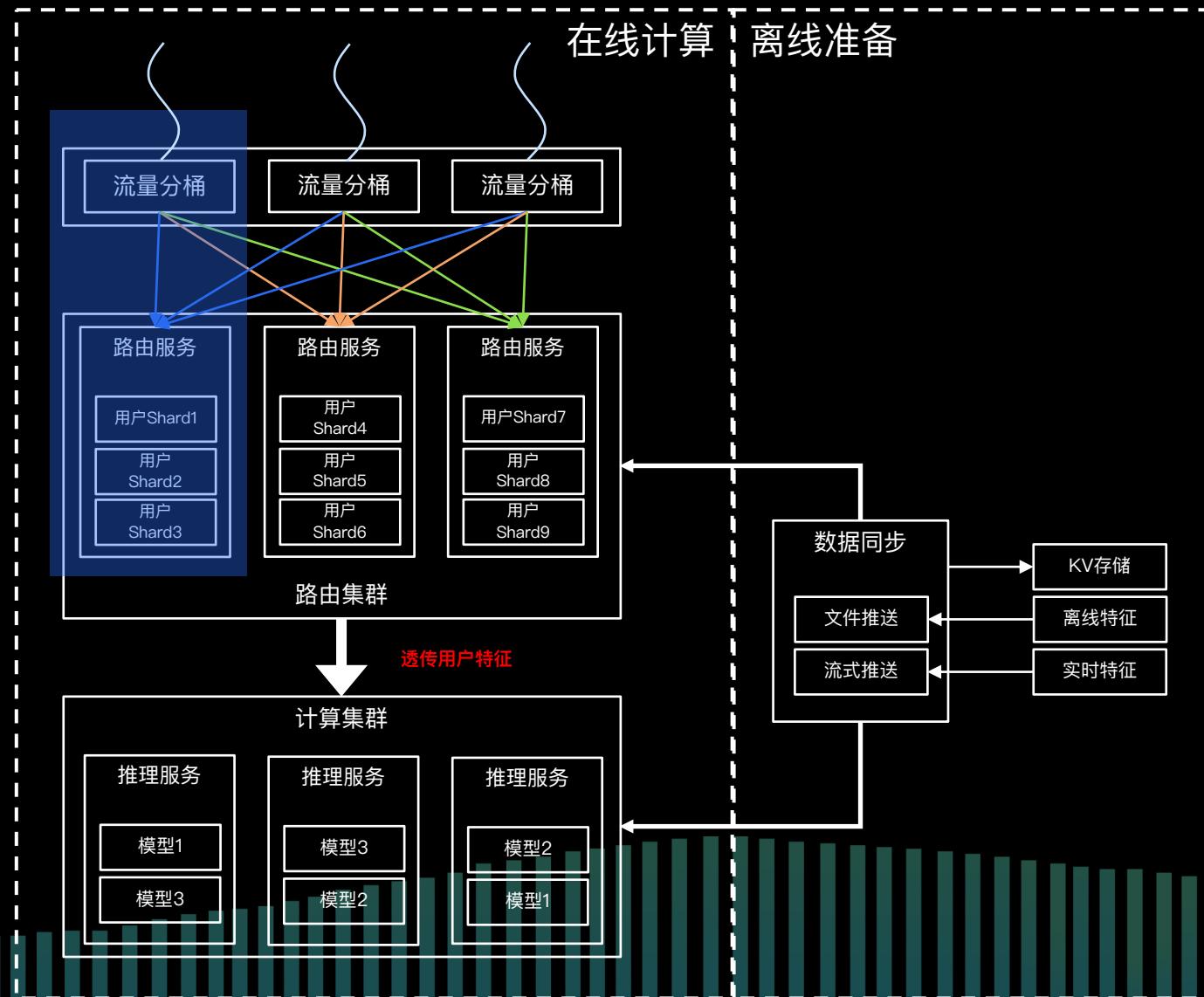
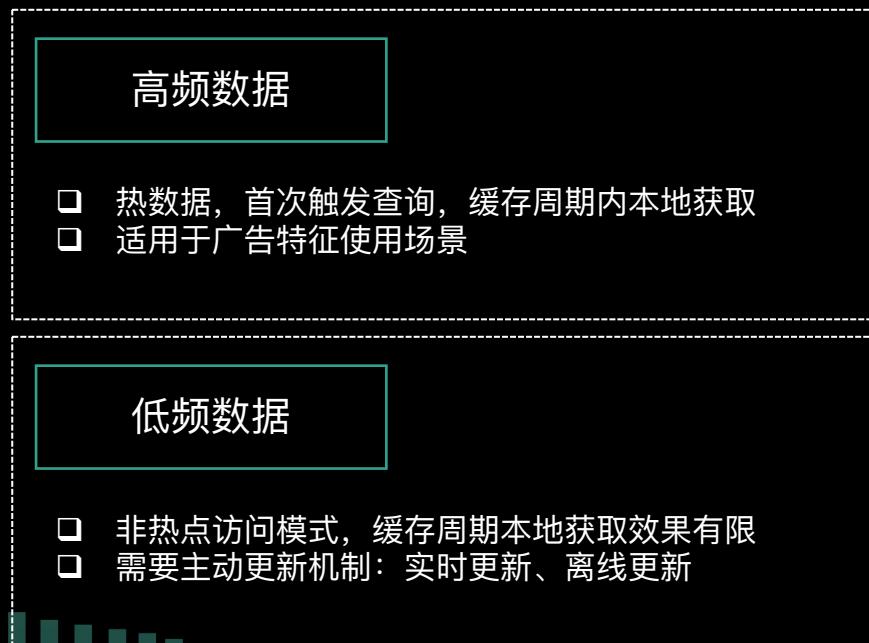
数据 – 用户特征查询长尾现象

- 用户特征服务特点
 - 引擎与存储跨IDC网络: N毫秒
 - 推荐引擎实时读取 (多列)
 - 用户特征KV实时更新 (多列)
 - 训练任务实时读取 (多列)
- 导致用户特征查询请求存在长尾开销
 - 数据读写路由依赖分布式缓存
 - 大量非实时任务读写影响
 - KV服务波动直接影响推荐服务性能



数据 – 用户特征分区本地缓存

1. 用户特征Sharding分区
2. 路由计算透传用户特征
3. 数据准备机制



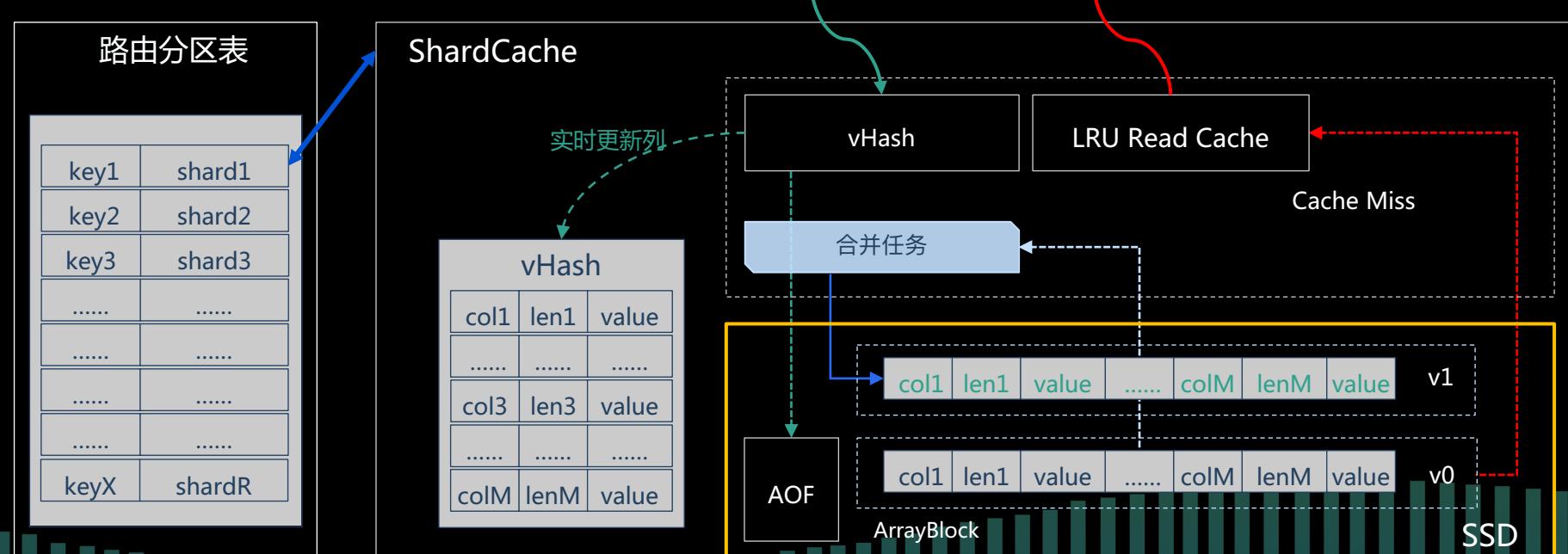
数据 – KV引擎优化

➤ 目标:

1. 提升性能, 优化读取长尾时耗
2. 成本优化, 避免全内存模式的高成本
3. 满足广告特征更新场景: 批量更新、增量更新

➤ 手段:

1. 引擎设计: 读写分离, 读缓存
2. 分层Tiered存储能力
 - ✓ SSD全列存储
 - ✓ 内存读热点数据全列缓存, 内存增量更 (部分) 新列存储
 - ✓ 异步合并更新, 异构硬件合并能力

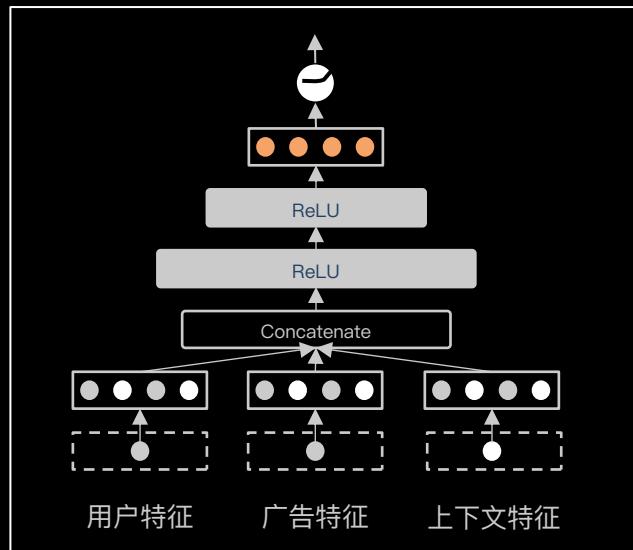


备注: 成本优化基于计算层缓存服务

模型 - 实时算力优化

深度模型计算

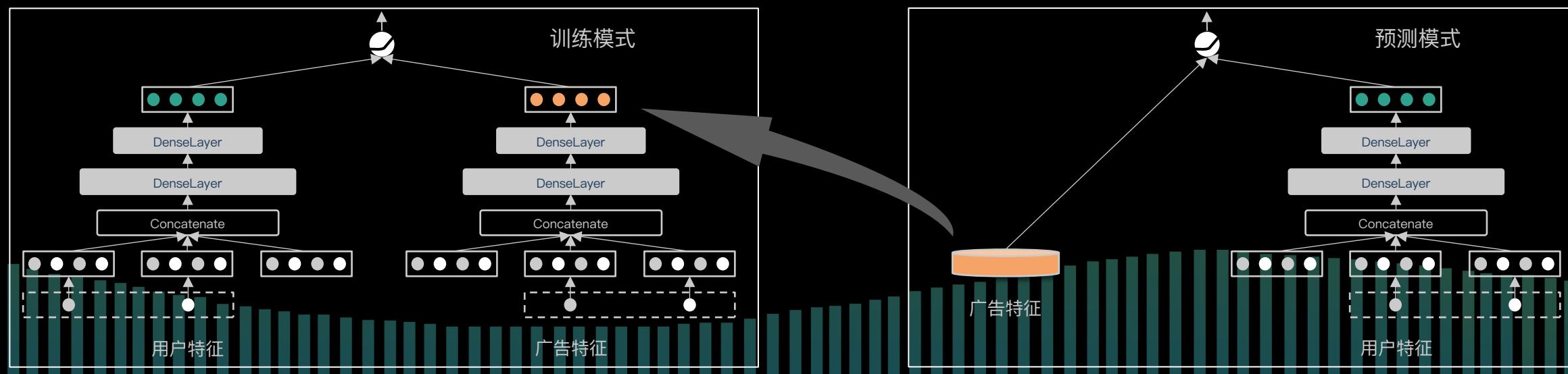
1. 需要实时计算全部网络，latency比较高
2. 计算时耗限制了模型深度与特征规模
3. 计算时耗与广告数量正相关
 - 无法适应长远发展



实时离线结合，提升系统整体算力

离线计算与在线计算相结合设计

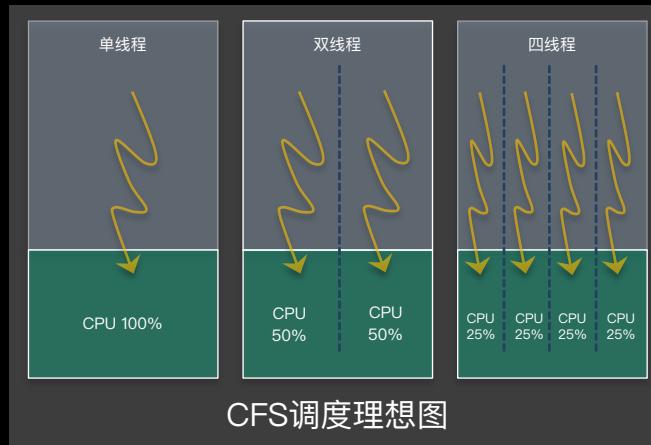
1. 离线批量计算广告子网络，在线查询广告子网络参数
2. 在线实时计算用户子网络
3. 在线深层模型：点积、全连接等



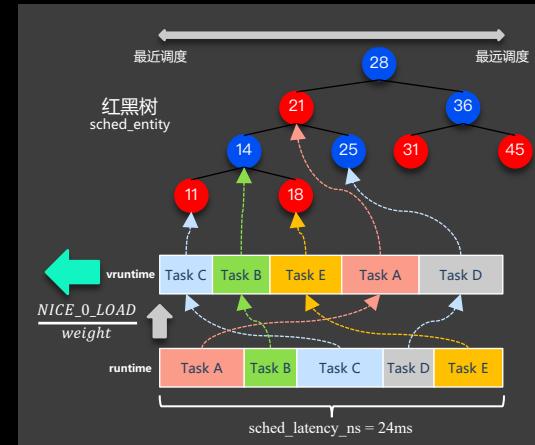
算力 – 任务调度长尾现象

1. 系统调度下，任务长尾达到13ms以上

| 长尾请求分析 | | | | | |
|----------------|------------|------------|------------|------------|-----------|
| 不隔离核心长尾请求各阶段时耗 | | | | | |
| time | seq | stage1(ms) | stage2(ms) | stage3(ms) | total(ms) |
| 15:43 | 44861773 | 13.518 | 9.657 | 0.149 | 23.326 |
| 15:43 | 44864425 | 2.094 | 19.425 | 0.309 | 21.830 |
| 15:43 | 44871774 | 2.456 | 0.000 | 25.969 | 28.426 |
| 15:43 | 44870221 | 2.514 | 0.000 | 24.859 | 27.373 |
| 15:43 | 4264371758 | 2.934 | 18.990 | 0.050 | 21.975 |
| 15:43 | 4264383022 | 12.894 | 11.093 | 0.054 | 24.043 |
| 15:43 | 182824230 | 3.051 | 17.037 | 0.065 | 20.155 |



CFS调度理想图



CFS调度策略： $vruntime = \text{权重占比} * \sum \text{调度到的时间片长度} = \frac{cpu.share}{cpu.share_{default}} \sum runtime$

2. 分析发现存在任务未及时唤醒：

- 闲时等待在 12ms+ 被唤醒

| 任务唤醒 50us 时耗 | | | | |
|--------------|-------|-------|------------|---------------|
| time | tid | pid | 任务唤醒延迟(ms) | thread_name |
| 20:15 | 12769 | 12719 | 12.223 | eal_thread_85 |
| 20:15 | 12769 | 12719 | 16.005 | eal_thread_85 |
| 20:15 | 12761 | 12719 | 16.489 | eal_thread_77 |
| 20:15 | 12762 | 12719 | 11.073 | eal_thread_78 |
| 20:15 | 12763 | 12719 | 13.602 | eal_thread_79 |
| 20:15 | 12764 | 12719 | 13.605 | eal_thread_80 |

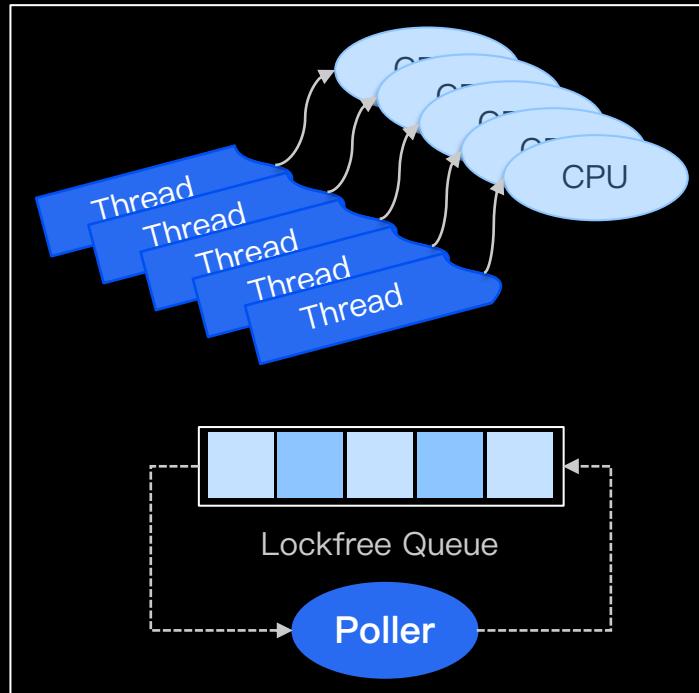
任务调度长尾



算力 – SN架构与用户态调度

思路：

1. 打破CFS公平调度原则
 2. 避免计算任务“等待事件”
- 计算资源分配
 - 计算任务独占大部分CPU时间片
 - 非计算任务共享少量CPU核
 - 计算任务调度模式
 - 不切换、不漂移、不睡眠
 - Shared Nothing计算架构
 - 用户态调度机制



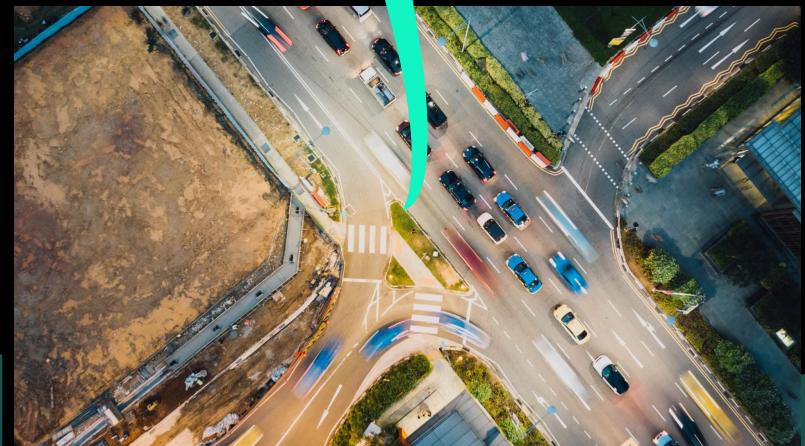
自研SN架构 & 用户态调度

3. SN用户态调度优化效果

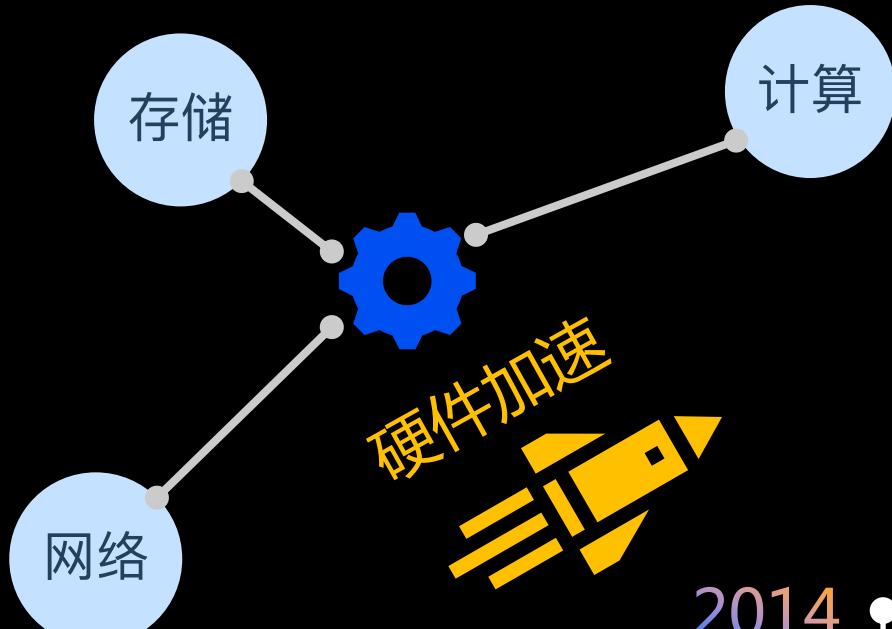
隔离核心以后长尾请求各阶段时耗

用户态调度后，任务长尾时耗降低明显

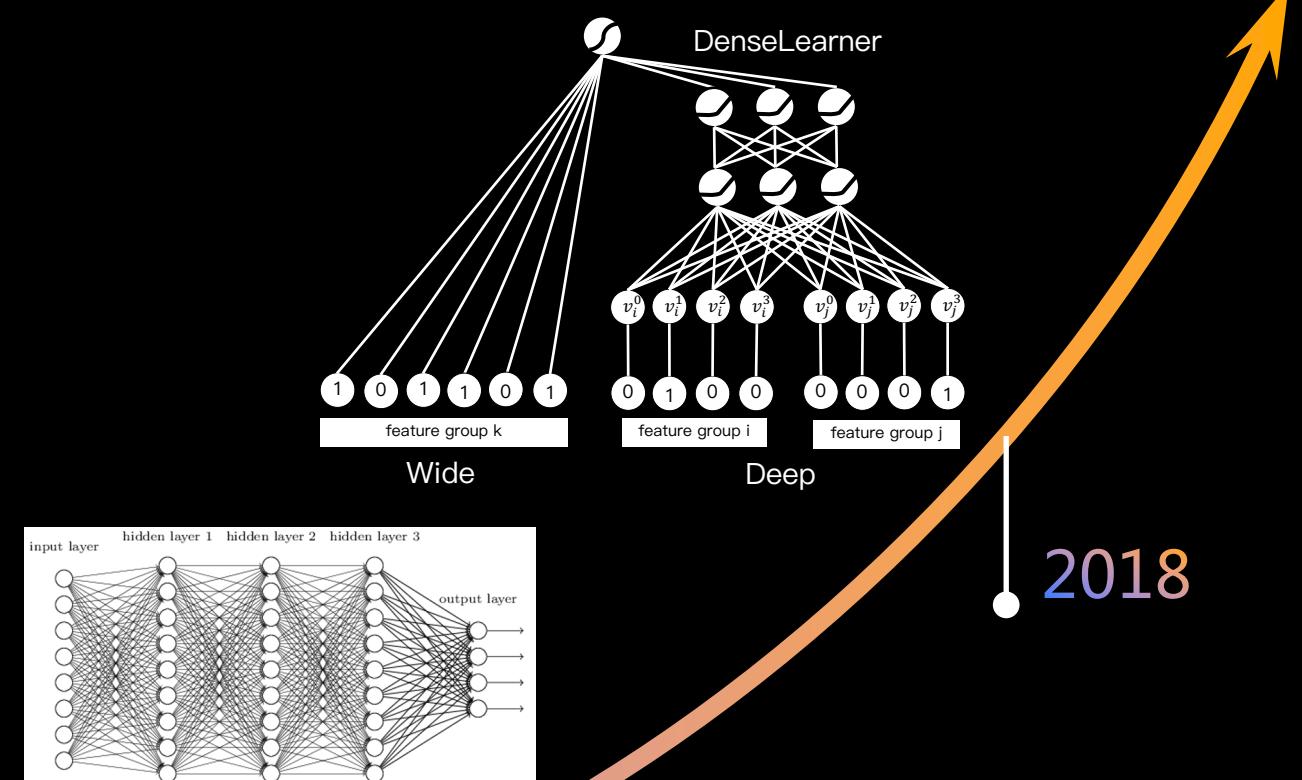
| time | seq | stage1(ms) | stage2(ms) | stage3(ms) | total(ms) |
|-------|------------|------------|------------|------------|-----------|
| 14:39 | 4026748448 | 2.276 | 18.728 | 0.045 | 21.050 |
| 14:39 | 4040943903 | 2.672 | 18.781 | 0.038 | 21.491 |
| 14:39 | 4011305509 | 3.191 | 17.164 | 0.036 | 20.392 |
| 14:39 | 3933165602 | 2.322 | 0.000 | 25.596 | 27.918 |
| 14:39 | 106869291 | 3.712 | 17.012 | 0.037 | 20.764 |
| 14:39 | 4013534501 | 3.566 | 17.220 | 0.042 | 20.829 |
| 14:39 | 4144218956 | 2.664 | 18.811 | 0.041 | 21.516 |



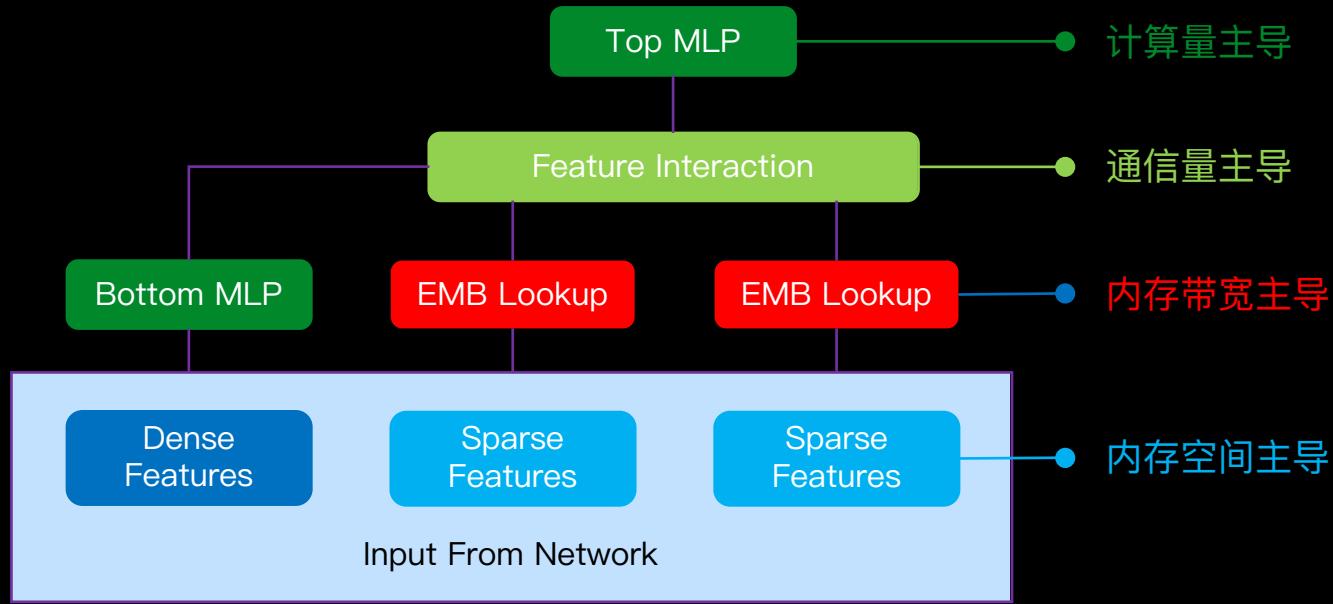
算力追求永无止境



$$y(X) = w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^{n-1} \sum_{j=i+1}^n w_{ij} x_i x_j$$



硬件加速 – 模型计算演进



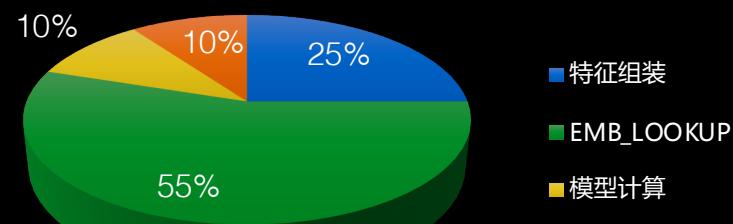
➤ 特征抽取、参数查询

- 各占转化率计算总时耗 约30%，共约60%
- 特征/参数的数量上一个量级（对比点击率计算）

➤ 深度模型计算量

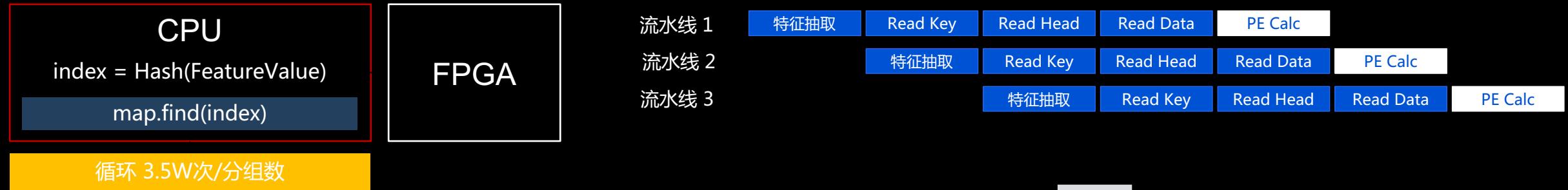
- Embedding维度增加
 - 128、256、.....
- 追求更高效，参数精度提升
 - Bfloat16、Float、double、.....

转化率预估服务时耗分布

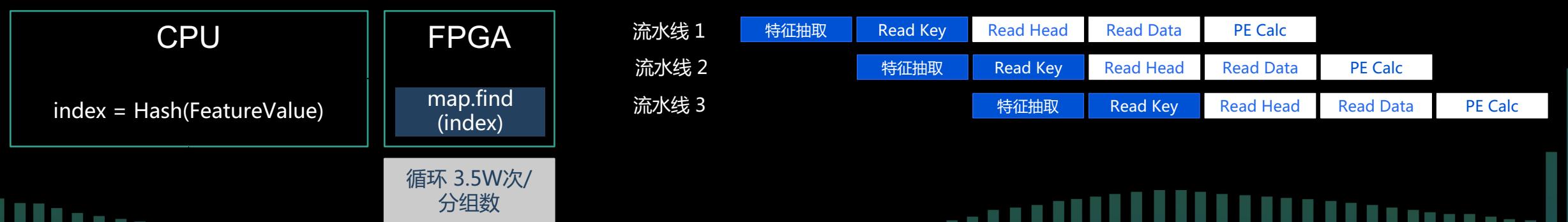


导致计算量成倍数增加，时耗超出预期

硬件加速 – 异构PIPELINE计算

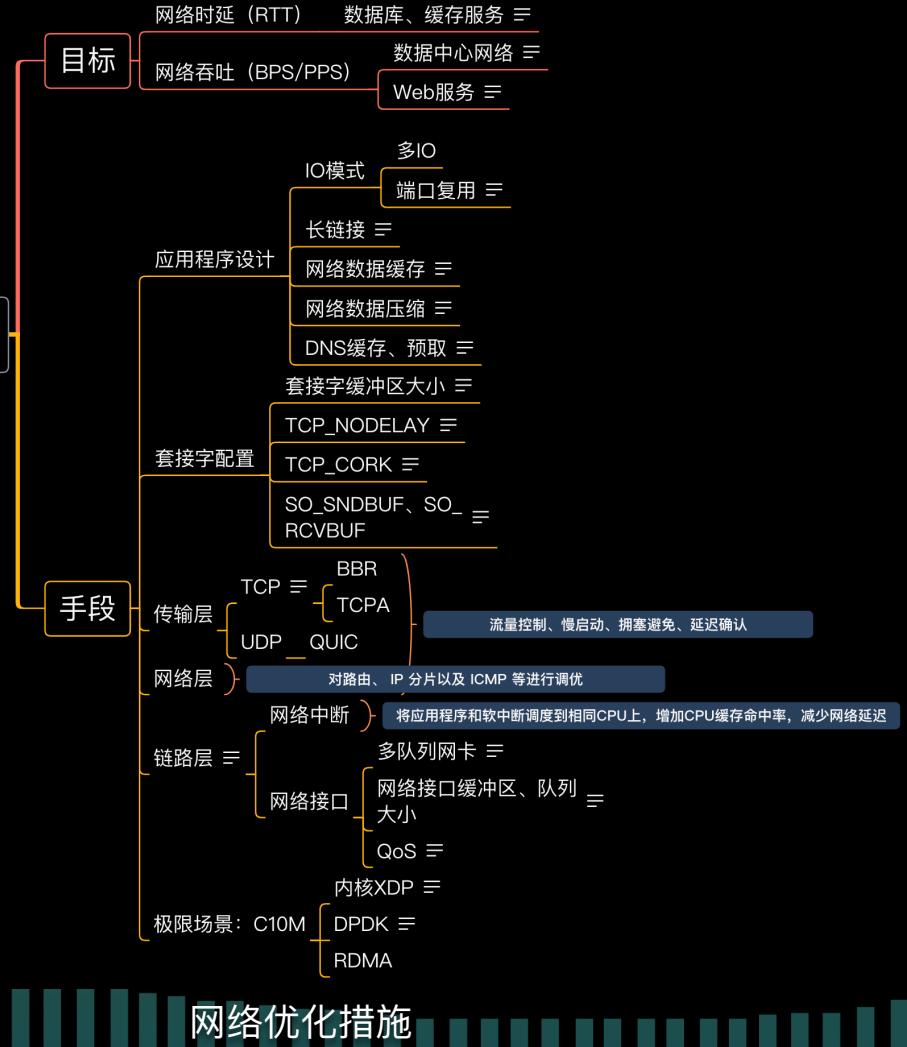


- FPGA实现“线性探测+开链结构”哈希表，加速查表性能
- 利用FPGA多PE计算单元，充分发挥Pipeline模式的能力



硬件加速 – 网络优化概述

网络性能优化



C10M特点

- 千万级并发连接数（百万连接/秒，每个连接持续约十秒）
- 10、50、100Gb/秒的传输速度
- 10微秒的延迟、10微秒的抖动

内核是问题所在

优化数据面性能

Kernel ByPass

- 数据包直接传递到业务逻辑

优化Cache利用率

- 线程核间绑定、无锁设计、避免切换开销

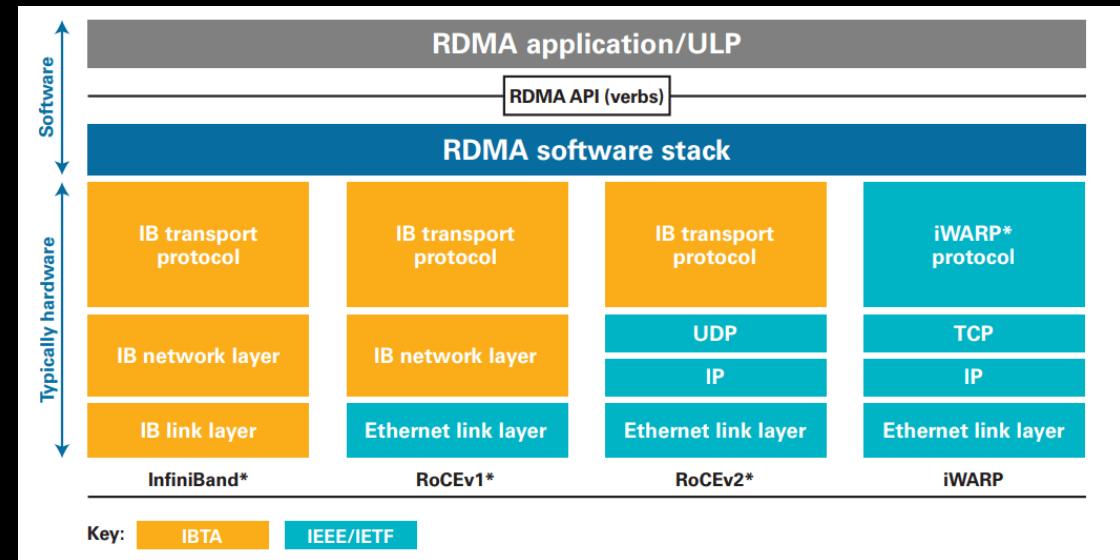
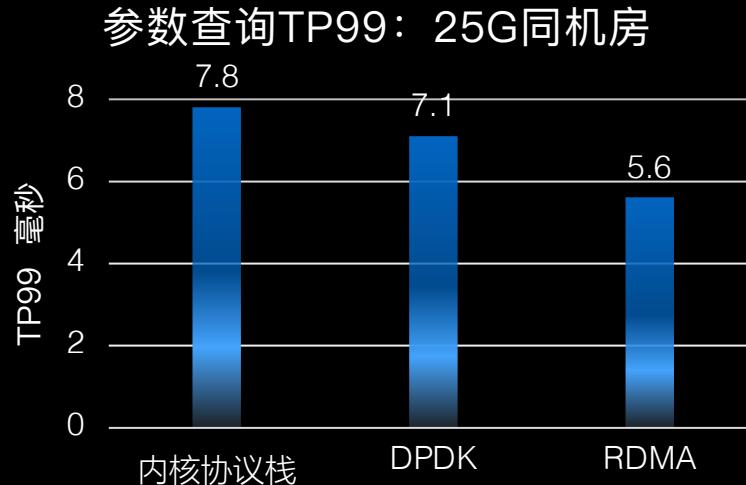
优化内存使用开销

- 网络传输所需的内存预留
- 大页内存减少地址转换性能开销
- 访问同一NUMA节点，减少内存远端访问

IRQ优化

- 多网卡队列核间绑定，均衡中断开销

硬件加速 – KERNEL BYPASS

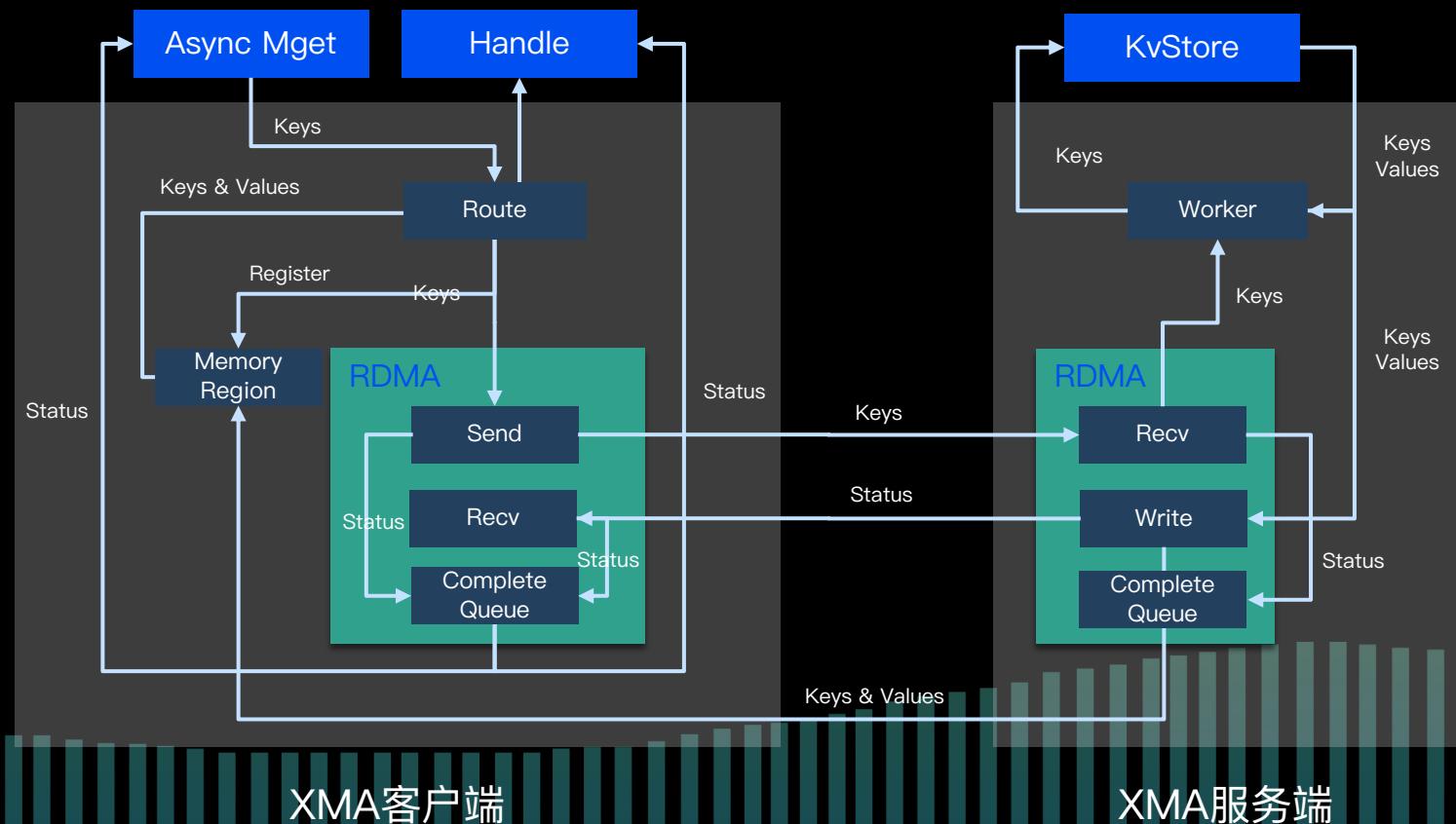
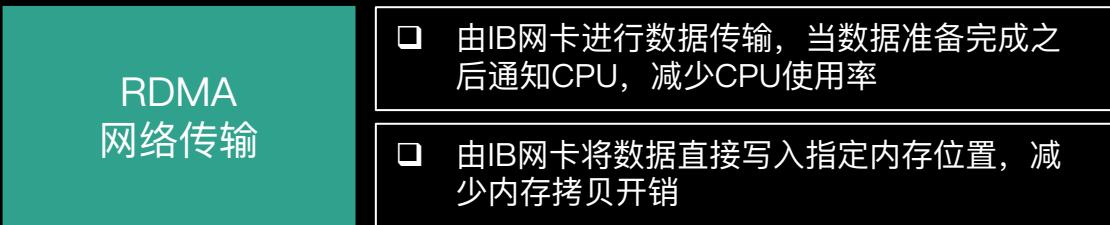


- RDMA
 - 核心技术: 协议栈硬件Offload
 - 减少中断和内存拷贝、降低时延、高带宽
 - 需要特定网卡支持
 - 引入额外开发成本
 - 不同于传统网络编程的API

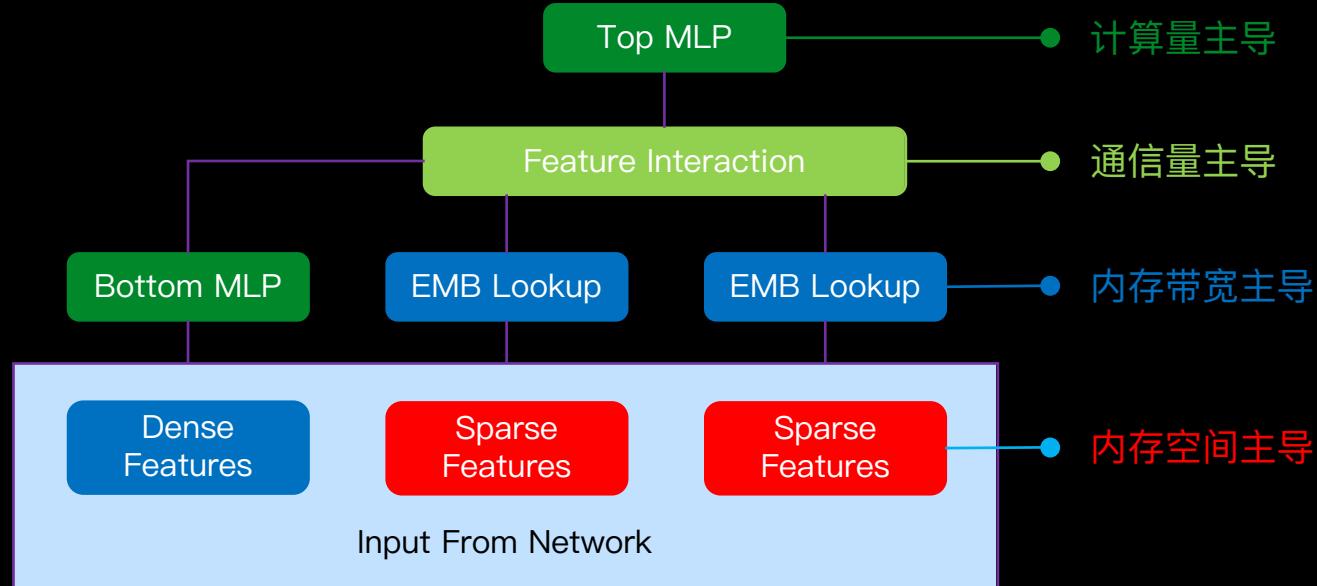
| | InfiniBand | iWARP | RoCE |
|------|------------|-------------|-------|
| 标准组织 | IBTA | IETF | IBTA |
| 性能 | 最好 | 稍差 (受TCP影响) | 与IB相当 |
| 成本 | 高 | 中 | 低 |

硬件加速 – XMA网络框架

- 沉淀网络框架XMA
 - 屏蔽TCP、RDMA编程接口差异，统一API
 - 自适配硬件差异，业务不感知网络通道模式



硬件加速 – 模型突破单机内存大小



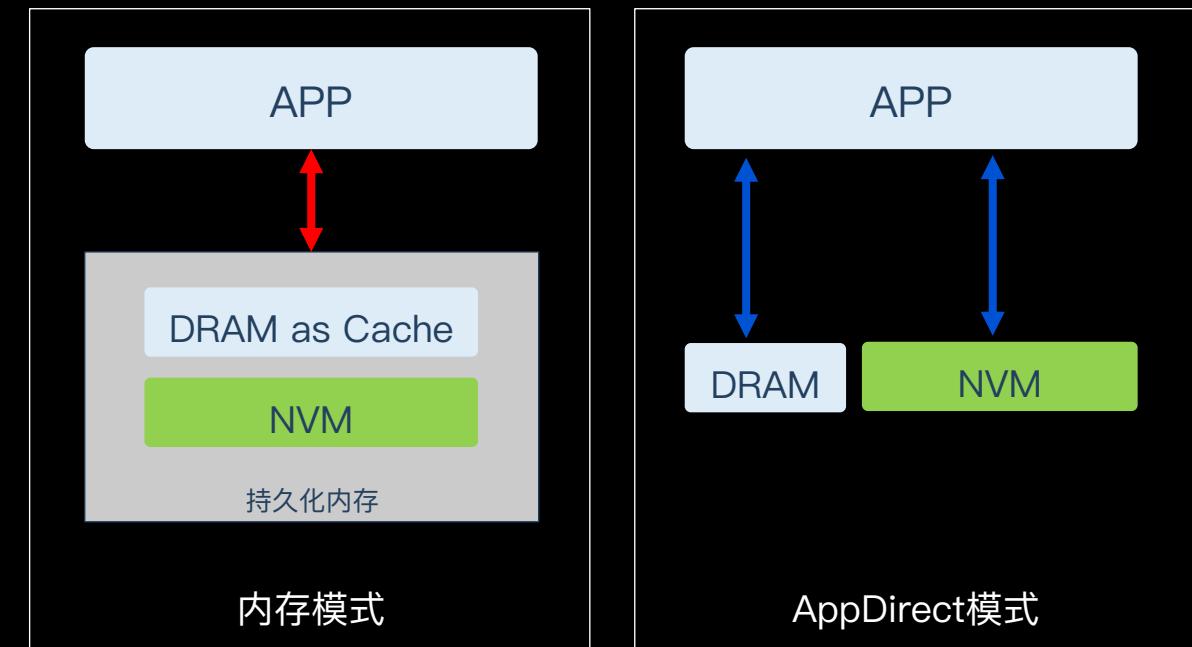
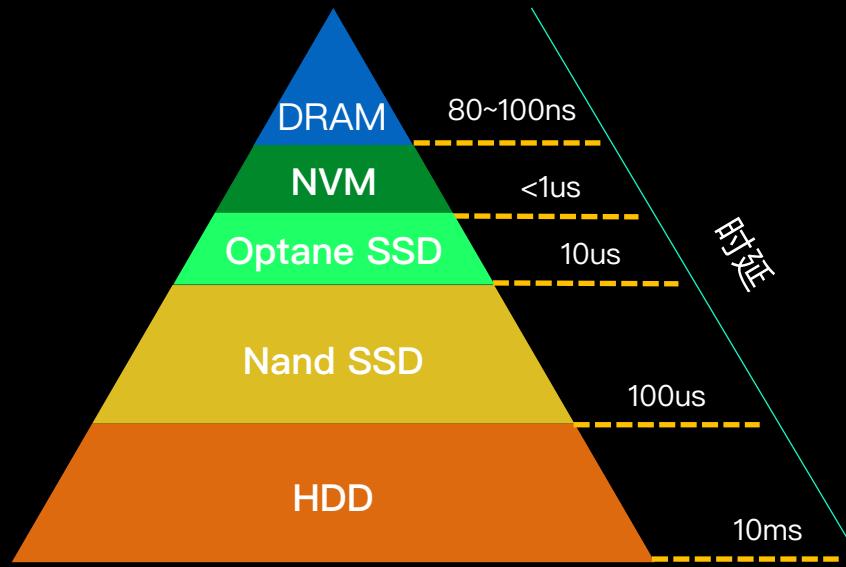
- 十万亿维度稀疏特征
- 单模型达到10T级别
- 在线预估带来新的挑战
 - 单模型的参数存储集群规模增加两个量级
 - 查询百、千级参数存储服务，SLA受高扇出影响

| 服务扇出量与SLA的关系 | |
|--------------|--------|
| 节点数 | SLA |
| 1 | 99.99% |
| 100 | 99% |
| 1000 | 90% |

降低在线参数服务器集群规模，提升参数请求SLA

硬件加速 – 持久化内存

- NVM – 持久化内存
 - 基于3D Xpoint技术的非易失性“内存”
 - 容量大、速度快、成本低



硬件加速 – 持久化内存

- NVM特性

- 读写速度快：读延迟为DRAM的1-3倍，写延迟相当
- 单机容量大：比DRAM更廉价的扩容方案，单机最大支持2个Socket，共8TB
- 持久化能力：机器重启数据不丢失，数据恢复速度高，提高系统可用性
- 成本低：单GB折合TCO约为内存 30%

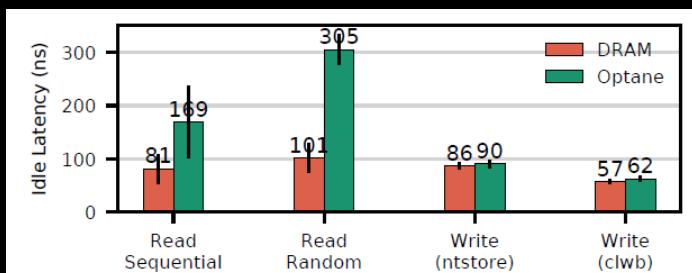


Figure 2: **Best-case latency** An experiment showing random and sequential read latency, as well as write latency using cached write with `clwb` and `ntstore` instructions. Error bars show one standard deviation.

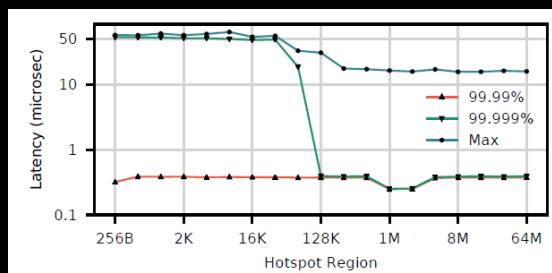
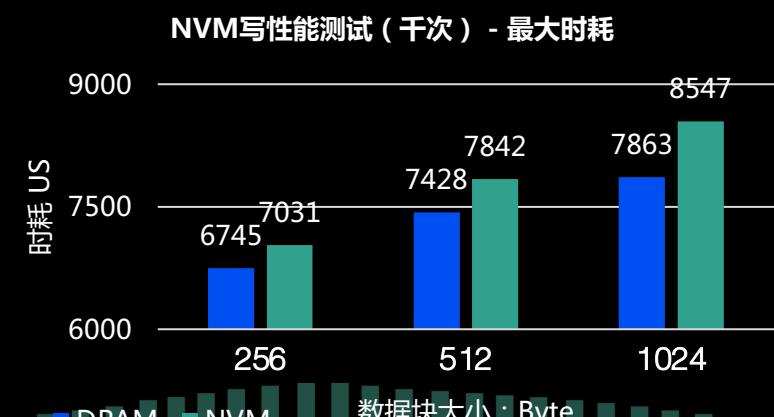
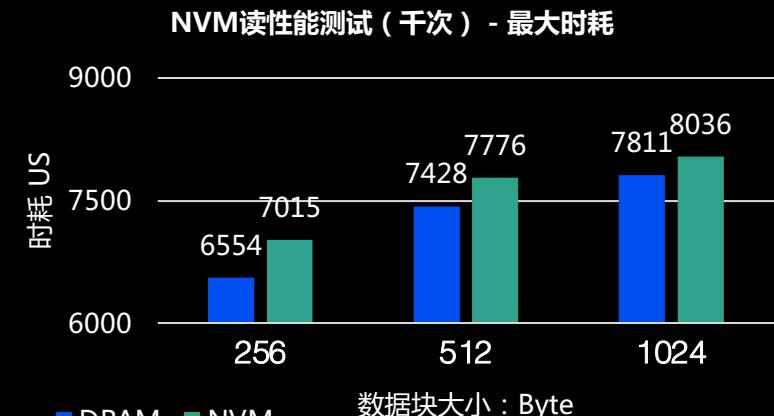


Figure 3: **Tail latency** An experiment showing the tail latency of writing to a small area of memory (hotspot) sequentially. Optane memory has rare “outliers” where a small number of writes take up to 50 μ s to complete (an increase of 100 \times over the usual latency).

- 256B的单“块”限制，单次读写数据达到256B以上，性能更好
- 更适合读多写少场景，访问带宽不对称



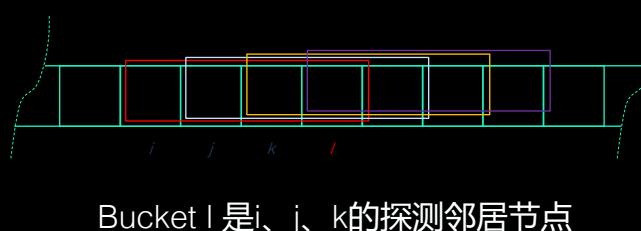
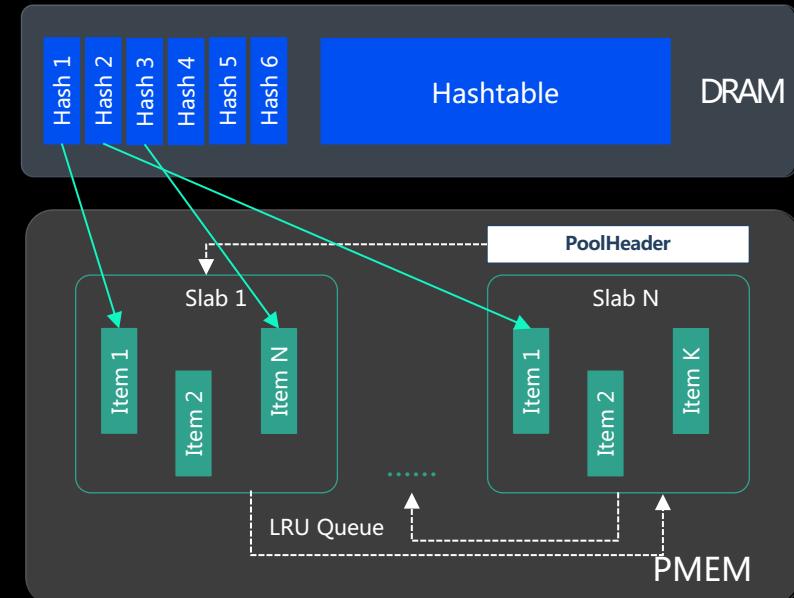
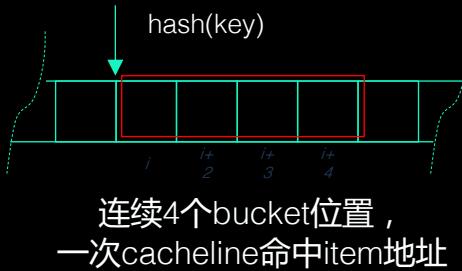
硬件加速 – 持久化内存引擎设计

在线参数服务器

- TB级模型全量缓存
- 批量写入后提供实时读取服务
- 优化在线读取长尾延时

PS存储引擎优化

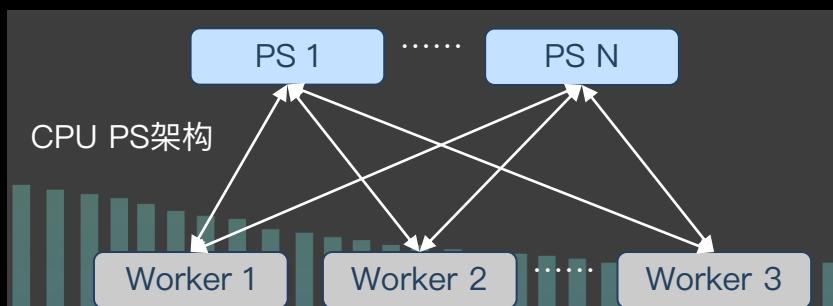
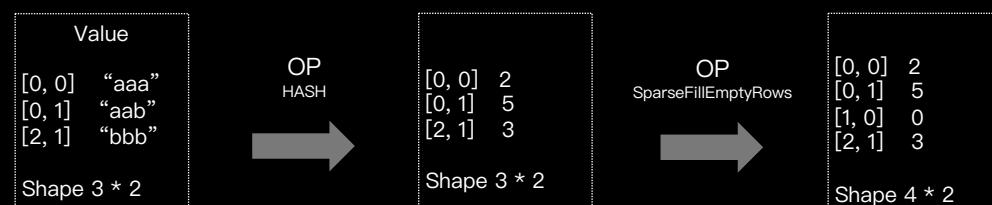
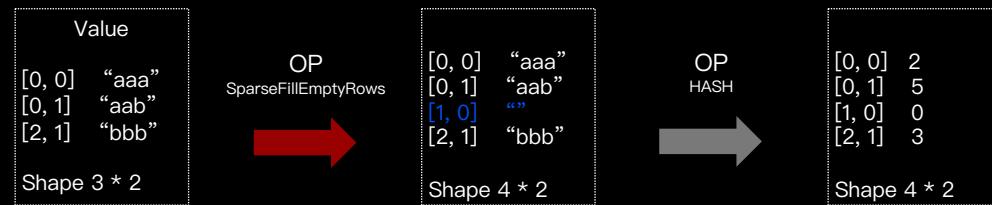
- 线性探测哈希表
- 内存索引，NVM大容量存储稀疏参数
- NUMA + NVM，亲和性绑定



硬件加速 – 训练加速

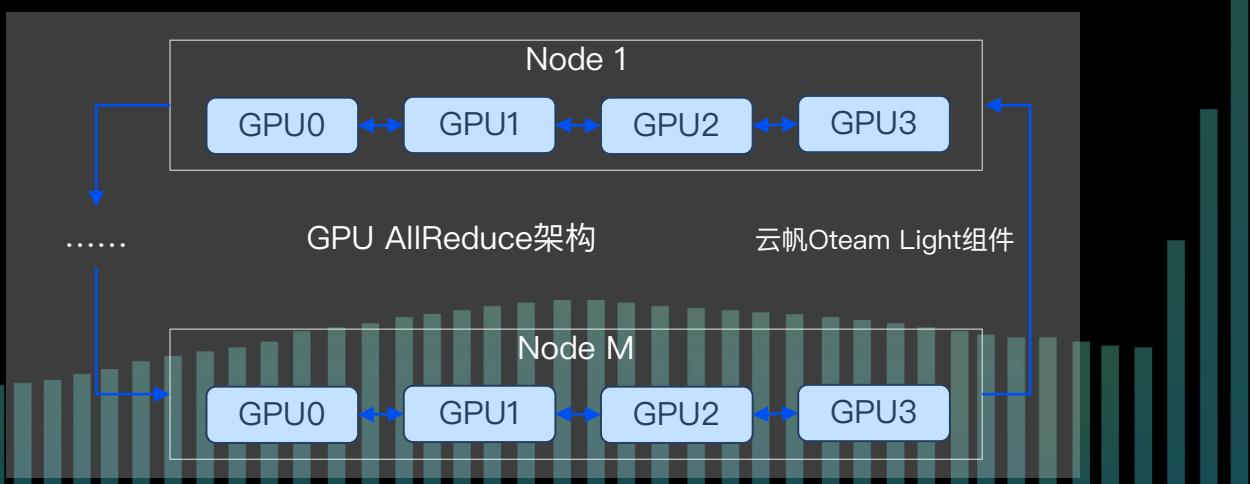
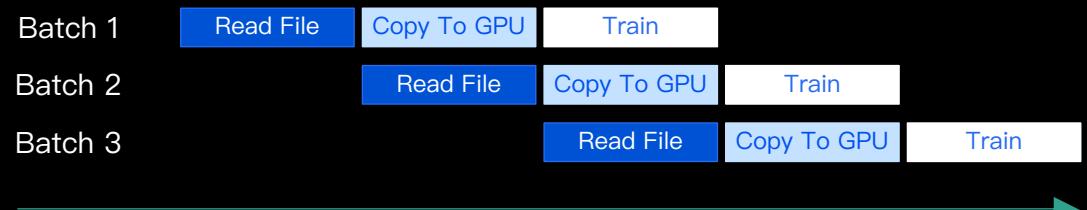
1. 优化网络结构:

- 优化稀疏特征的embedding lookup过程



2. 升级训练架构:

- Pipeline计算
- GPU加速: CPU算子优化迁移GPU
- 通信加速: AllReduce + RDMA



03

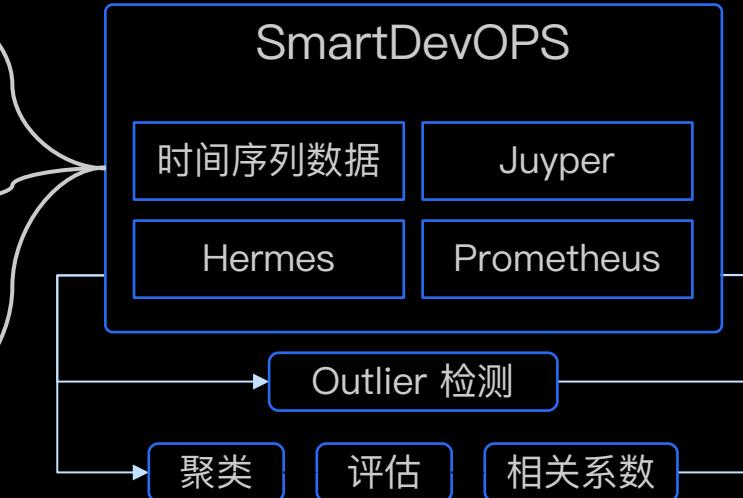
智能巡查调度优，保障服务质量

服务质量保障 – 现网服务质量巡查

业务指标 性能: 平均时耗、TP99
收入: 收入波动

运行参数 框架: ThreadNum、队列大小
算法: lookup 并发

基础设施 CPU: migrations、branch-misses
网络: rx_dropped、rx_missed_errors
内存: PageTables、Vm alloc Chunk



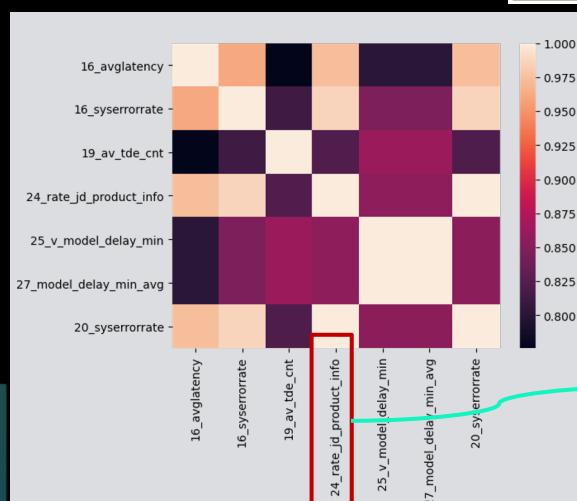
XX特征引发毛刺的数据分析: (322个监控指标)

特征过滤

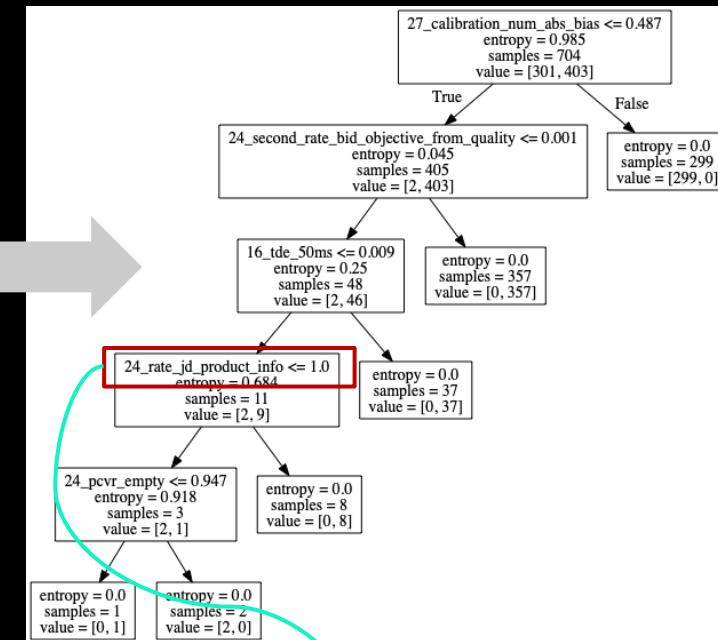
包装器特征选择

相关性计算

| 24_rate_XX_product_info | 20_syserrorrate | 16_tde_50ms | 17_failedrate | 21_avglatency |
|-------------------------|-----------------|-------------|---------------|---------------|
| 0.045579 | 0.002172 | 1717970 | 0.316139 | 6.447809 |
| 0.045581 | 0.001838 | 1723970 | 0.31638 | 6.466473 |
| 0.27641 | 0.4243 | 538300 | 0.006776 | 85.25079 |
| 0.27639 | 0.406945 | 536420 | 0.006654 | 85.46015 |
| 0.276406 | 0.412203 | 533590 | 0.006677 | 85.39333 |
| 0.276403 | 0.427223 | 534580 | 0.006805 | 85.63181 |
| 0.276405 | 0.419926 | 536090 | 0.006816 | 85.63678 |

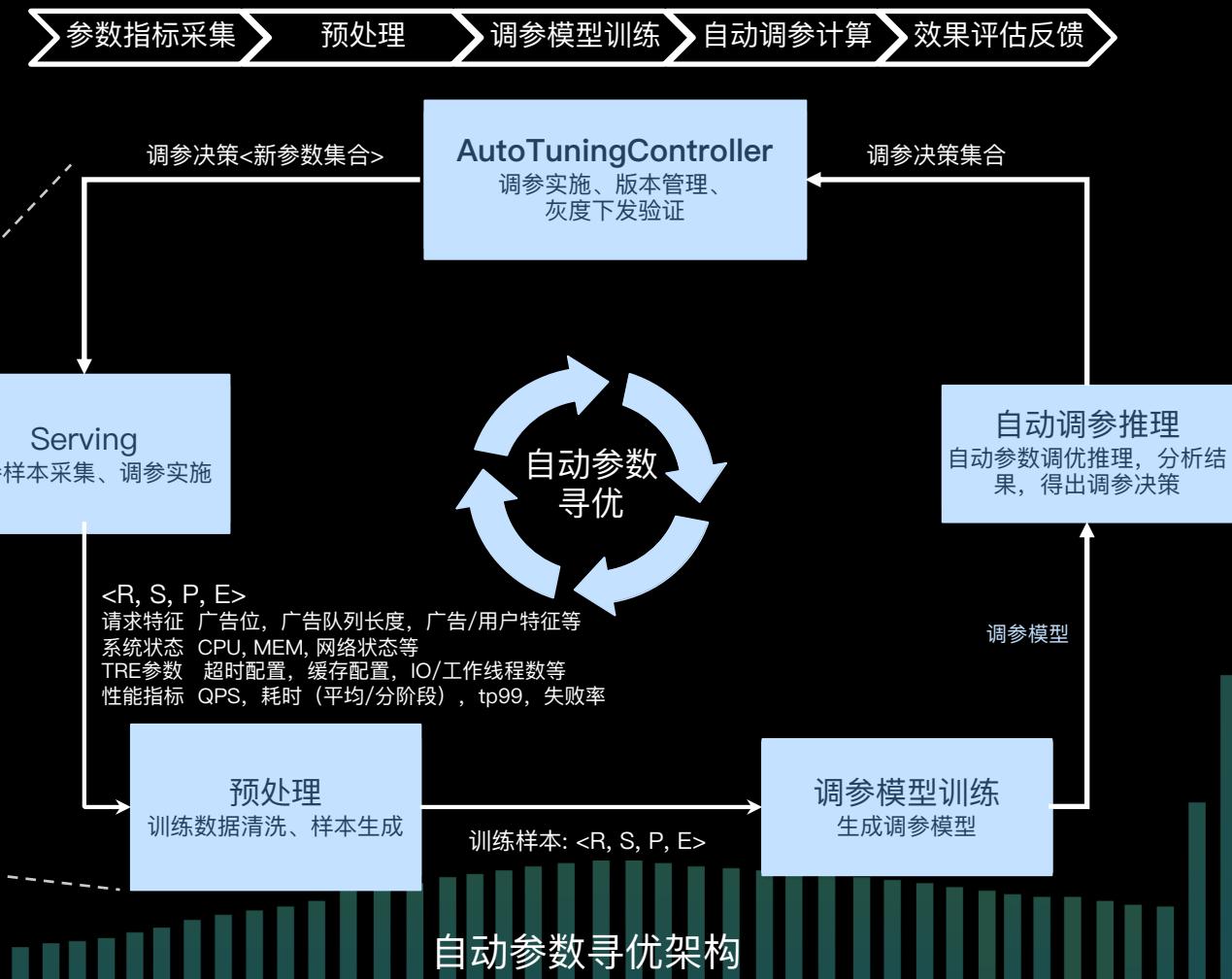
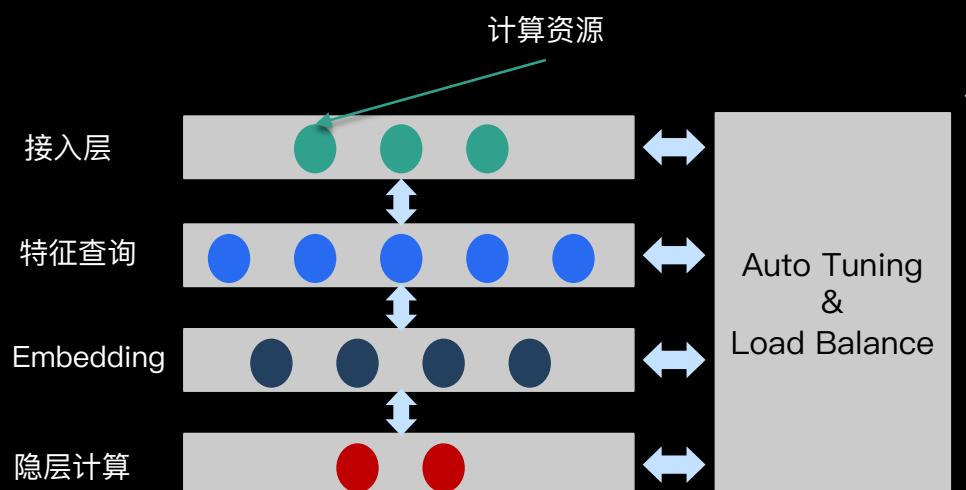


24_rate_jd_product_info
强相关
20_syserrorrate

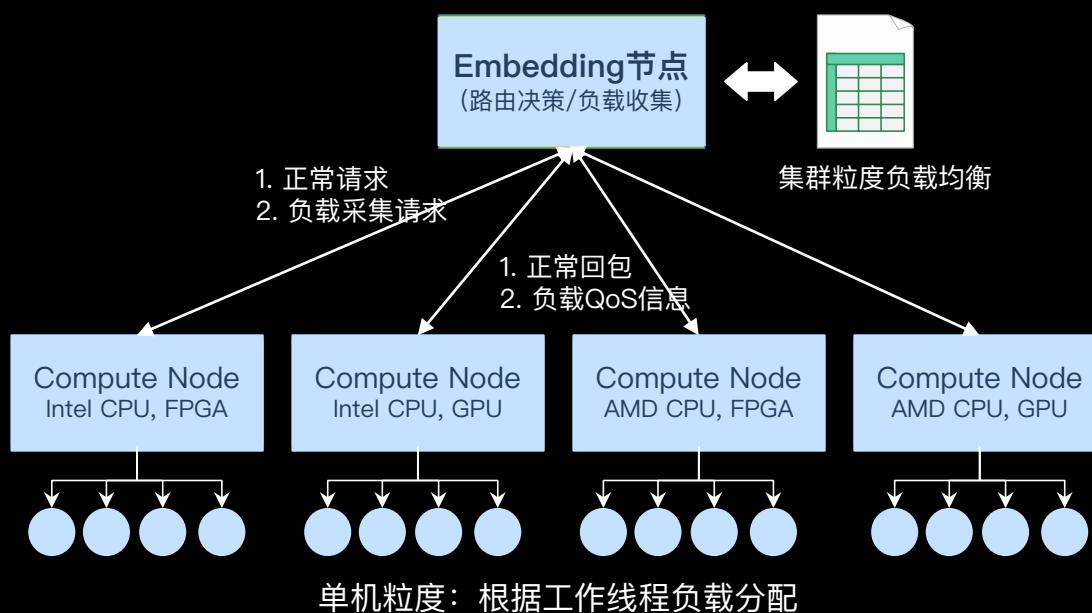


服务质量保障 – 运行状态调优

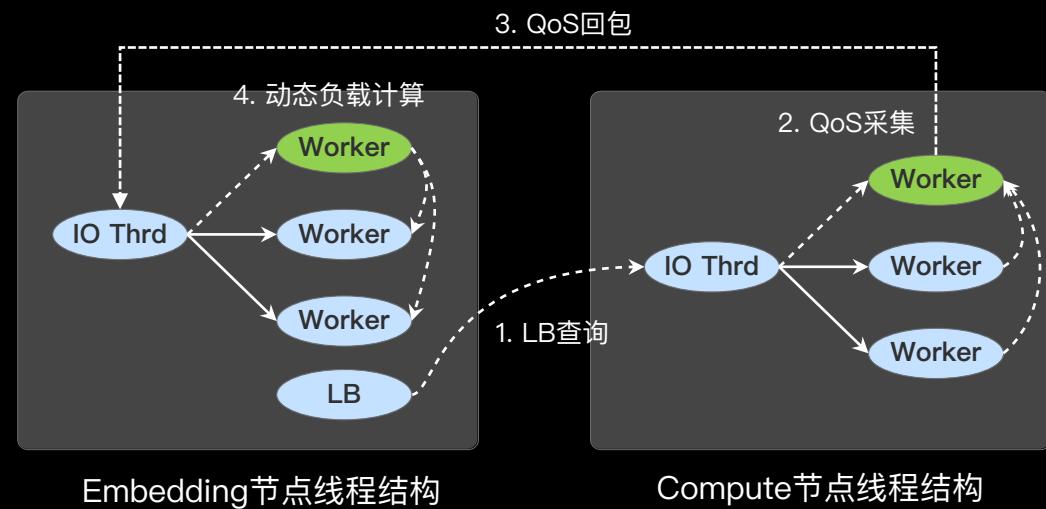
- 全局分层资源调度
 - 结合广告各流量场景，自适应调度各层资源配置
- 自动参数寻优
 - 集群粒度：计算资源自适应分配、分层负载均衡
 - 单机粒度：异构设备运行参数寻优，过载止损保护



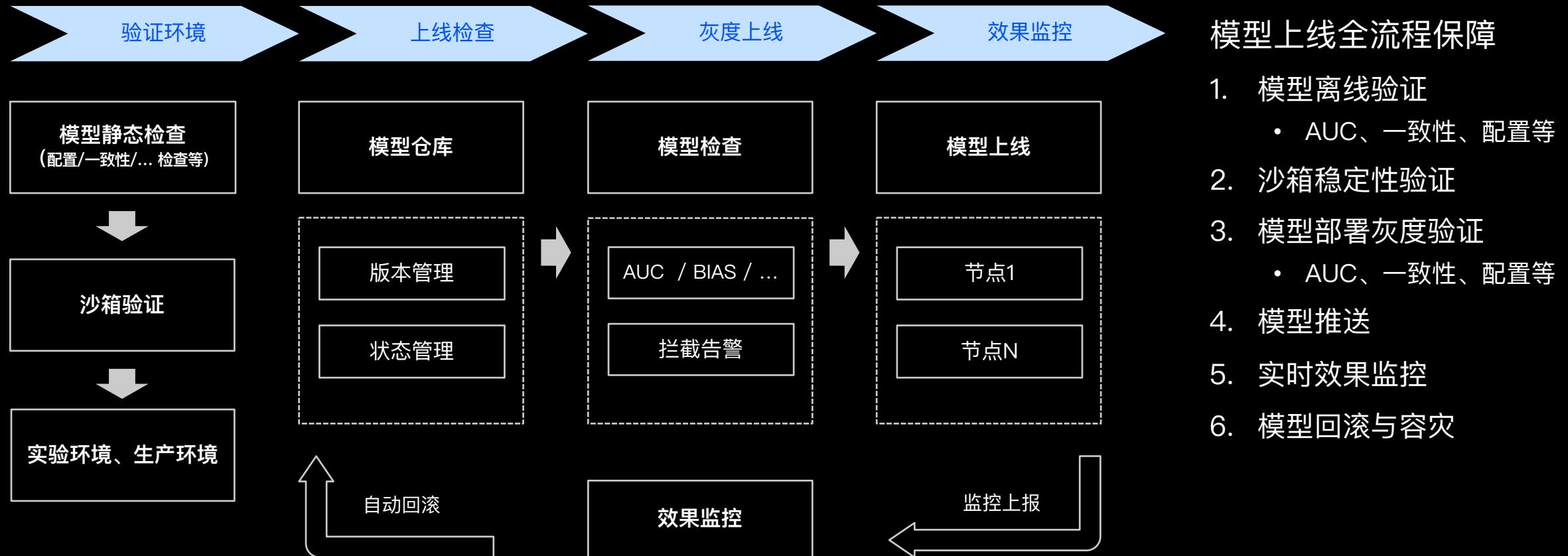
服务质量保障 – 动态负载均衡



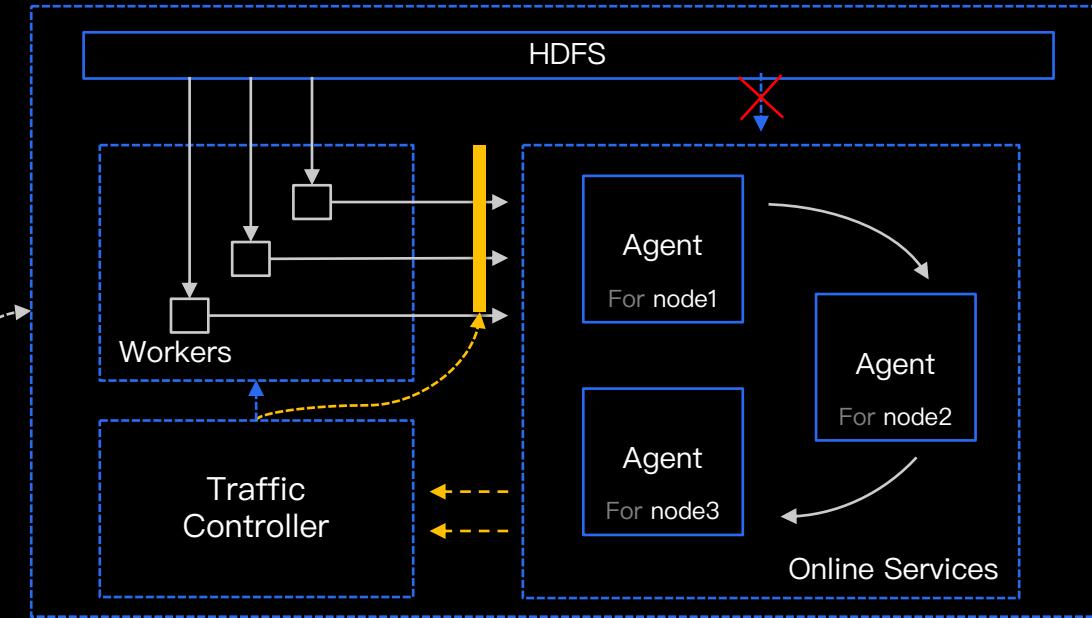
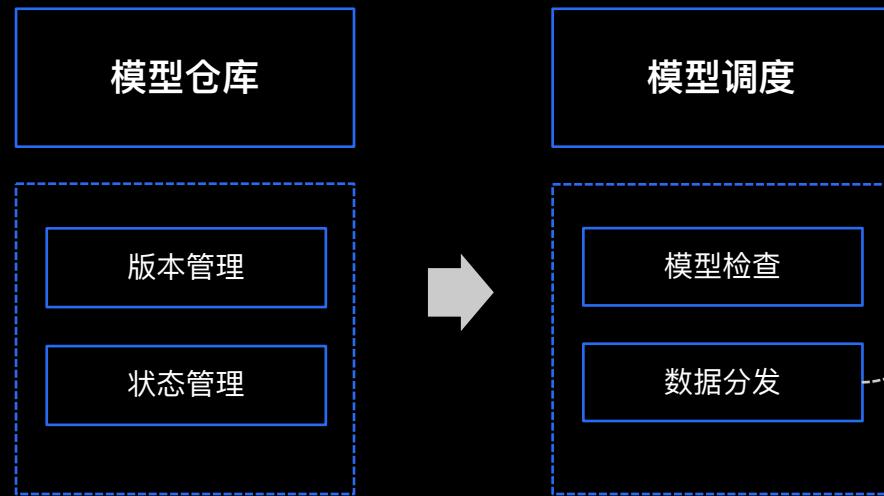
- 通过动态负载均衡机制，保障异构系统平稳运行
- 充分利用异构设备，负载与算力匹配
- 计算任务轻重分离，降低排队导致的长尾



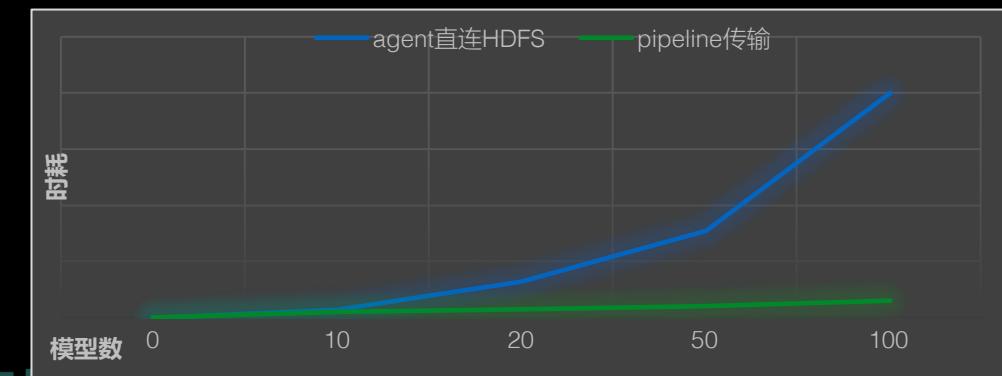
服务质量保障 – 模型调度系统



服务质量保障 – 模型调度链路优化



- Block粒度并行下载，对大文件下载效率提升5倍以上
- Pipeline传输，提升带宽使用和传输效率
- 自适应全局流量控制
- 模型传输前md5校验，分片crc校验，保证数据一致性



总结

软硬结合、追求性能的极致

➤ 两条腿走路

- 量化，立体化评估机制
- 优化，软硬结合，架构优化

智能巡查看优，保障服务质量

➤ 工程实践

➤ 引入AI加持

茹炳晟

腾讯TEG基础架构部资深技术专家



腾讯TEG基础架构部资深技术专家，业界知名实战派软件质量和研发工程效能专家，腾讯云最具价值专家TVP，中国商业联合会互联网应用技术委员会智库专家，畅销书《测试工程师全栈技术进阶与实践》和《高效自动化测试平台：设计与开发实战》作者，《软件测试52讲 - 从小工到专家的实战心法》专栏作者。

主办方：

Boolan
高端IT咨询与教育平台

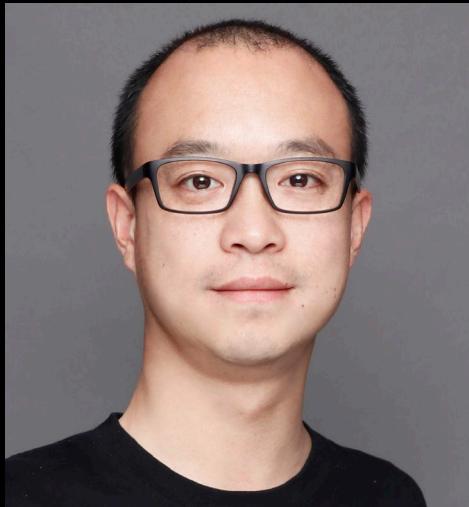
C++ Summit 2020

茹炳晟

腾讯 TEG 基础架构部

软件研发效能提升的 行业实践与探索

讲师简介

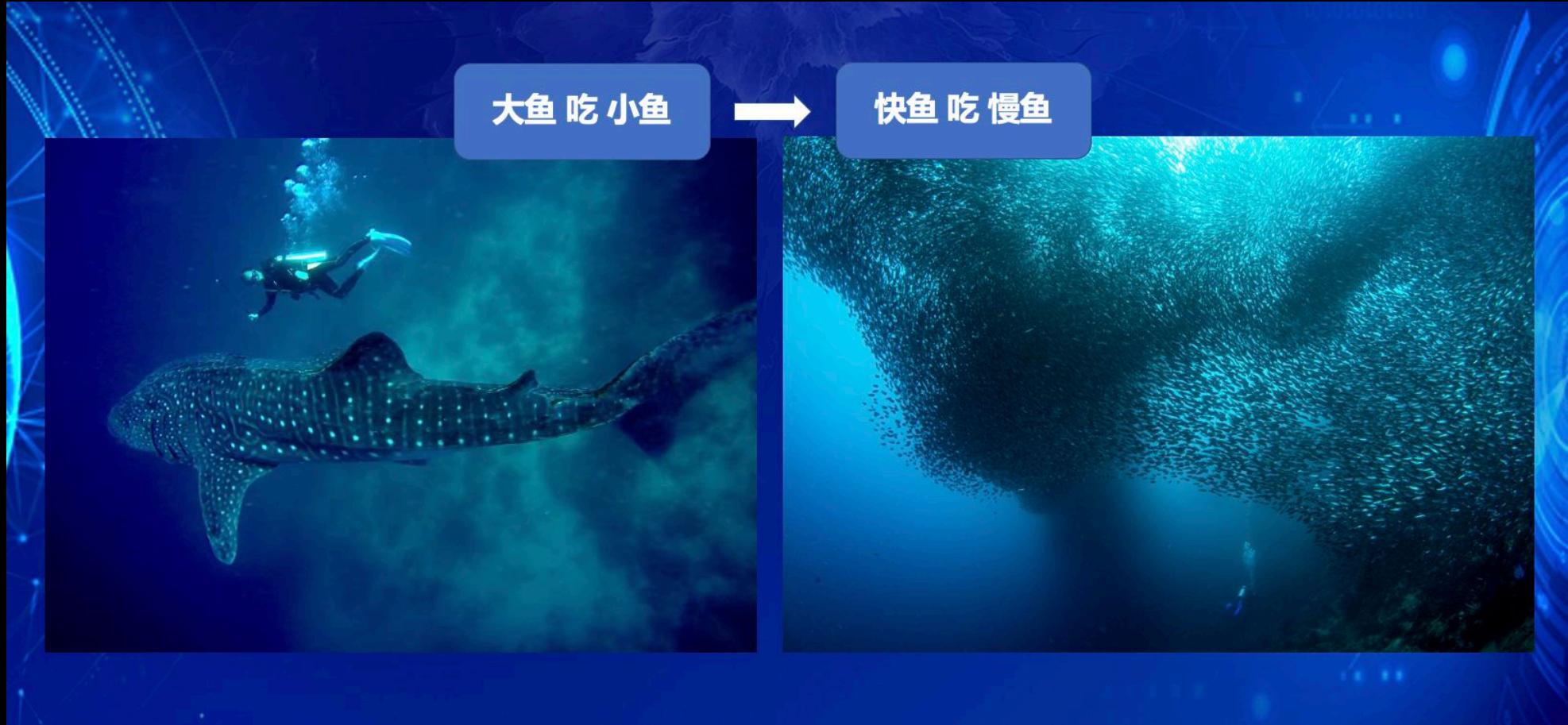


- 业界知名实战派研发效能和软件质量双领域专家
- 国内外各大技术峰会担任联席主席，技术委员成员和出品人
- 硅谷先进研发效能理念在国内的技术布道者
- 2020 年度IT图书最具影响力作者
- 2020 IT技术领导力年度互联网行业测试领域 技术专家
- 中国商业联合会 互联网应用技术委员会 智库专家
- 腾讯云最具价值专家TVP
- 畅销书《测试工程师全栈技术进阶与实践》作者
- 《高效自动化测试平台：设计与开发实战》作者
- InfoQ 极客时间《软件测试52讲-从小工到专家的实战心法》作者
- 新书《研发质量保障与工程效能》作者之一
- Certified DevOps Enterprise Coach

我们聊点啥？

- 研发效能的5大灵魂拷问
- “研发效能”的点点滴滴
- 研发效能提升的经验分享（8大原则）
- 研发效能的发展方向与未来展望

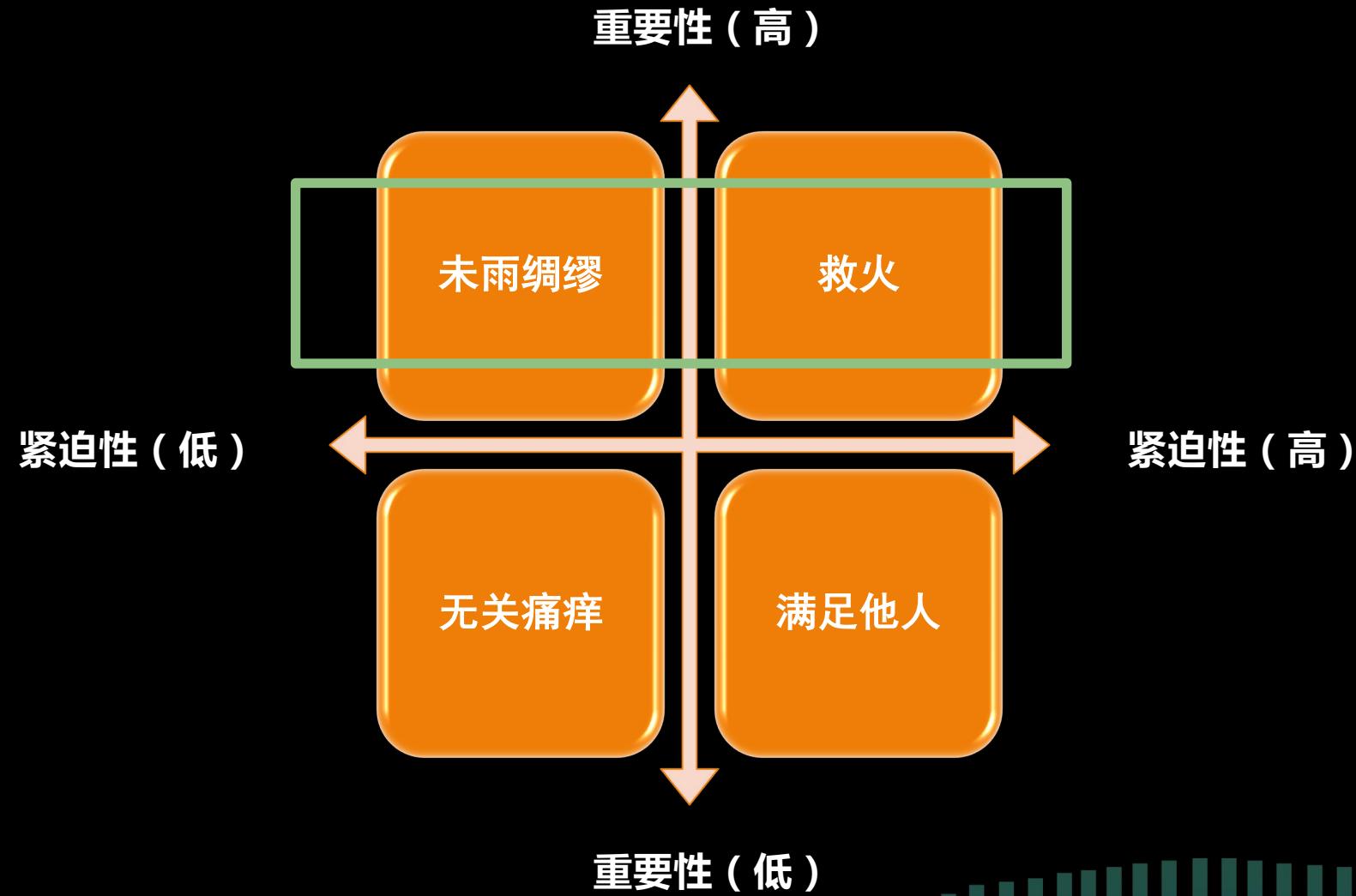
时代变了，底层逻辑也变了



研发效能的五大“灵魂拷问”



灵魂拷问1 - 研发团队的忙碌能代表高效率吗？



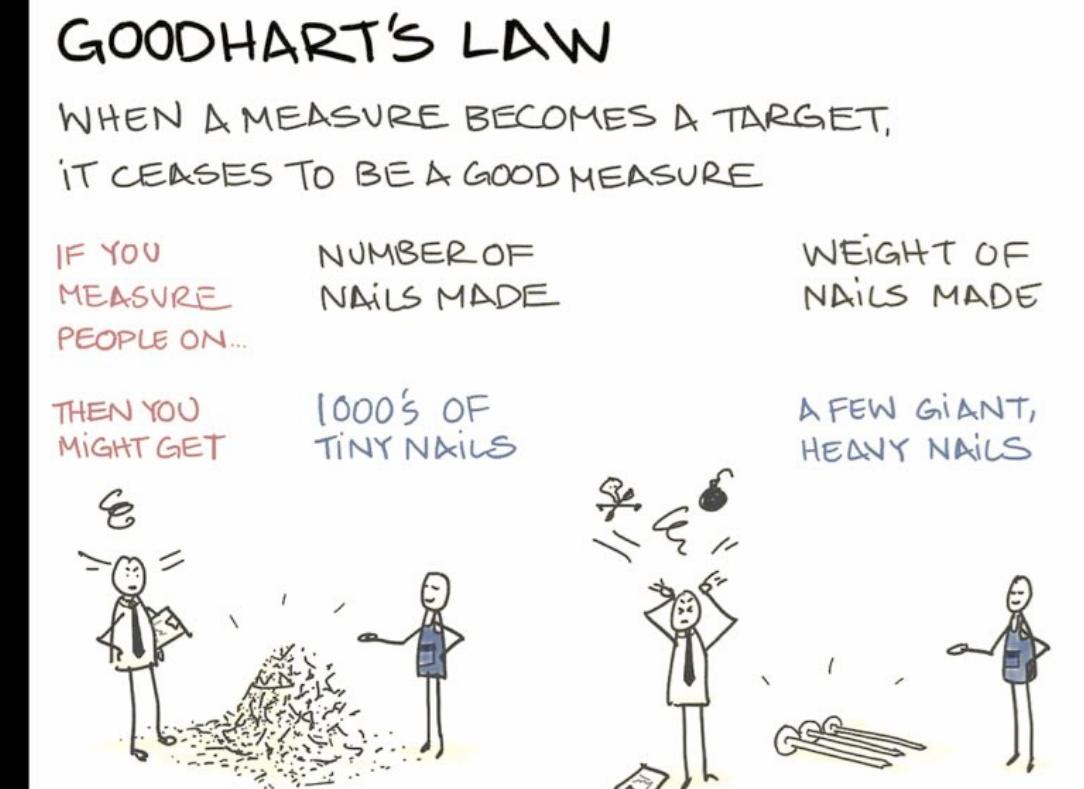
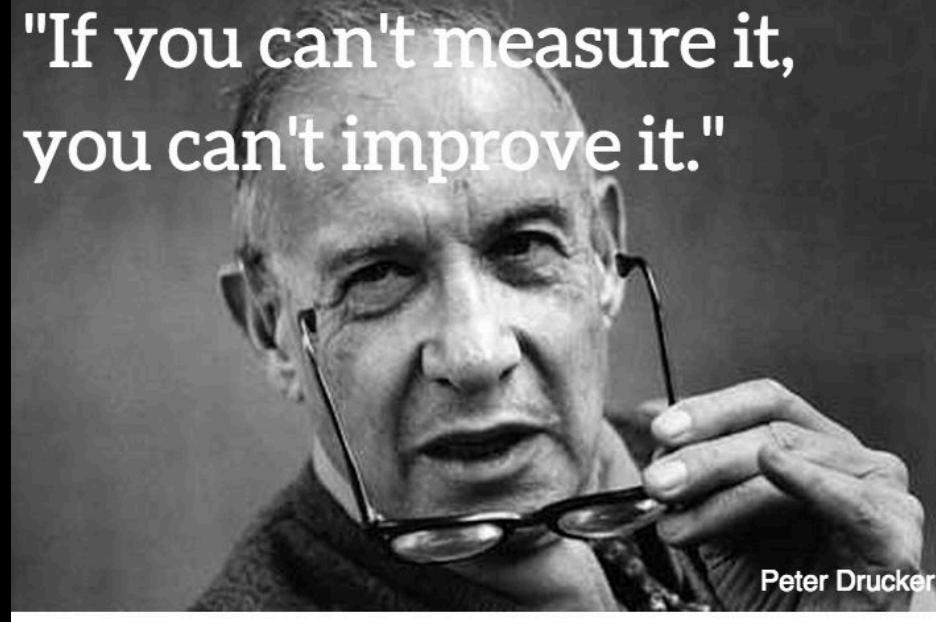
灵魂拷问2 - 敏捷是研发效能提升的银弹吗？



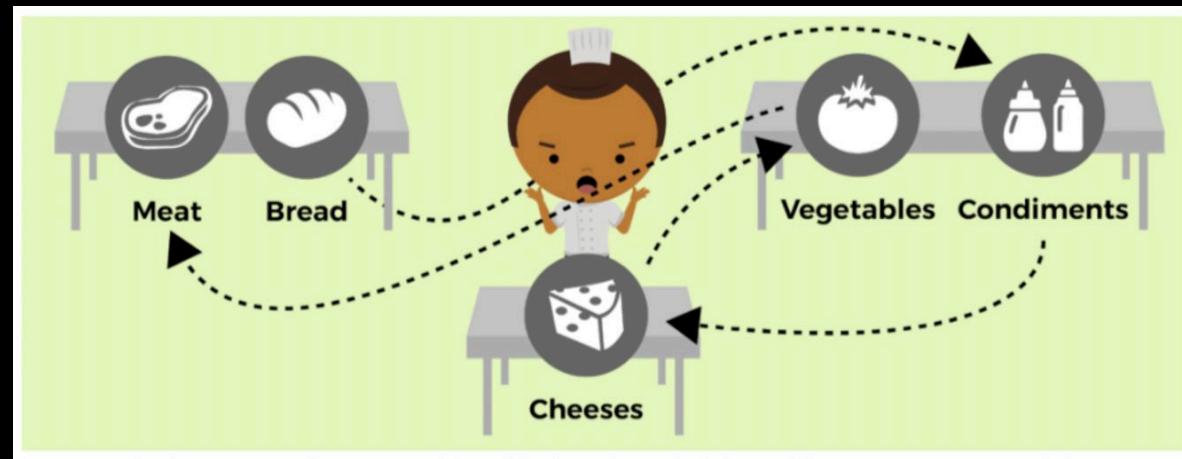
灵魂拷问3 - 自动化测试真的提升软件质量了吗？



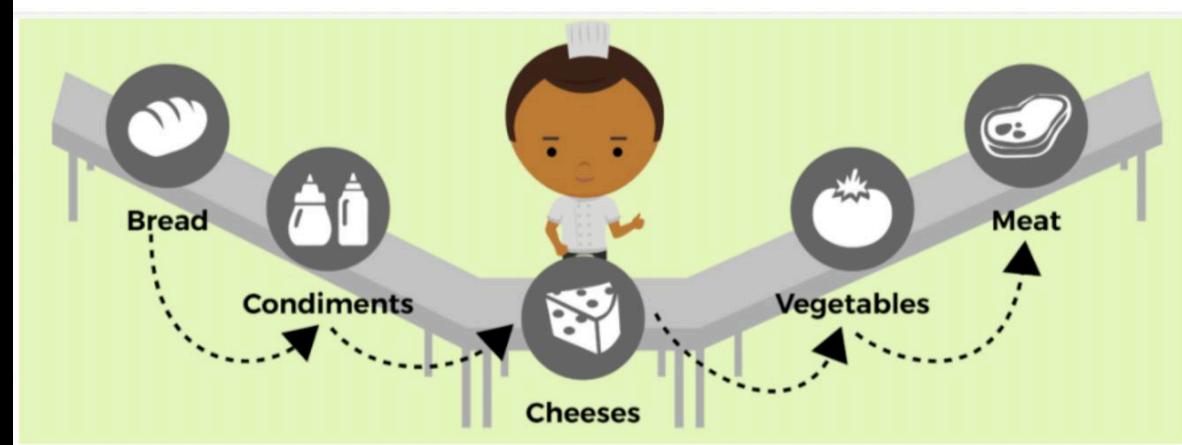
灵魂拷问4 – 没有度量就没有改进，这是真的吗？



灵魂拷问5 - 研发效能的提升一定是由技术驱动的吗？



Before Waste is removed in a kitchen (Sandwich making process example).



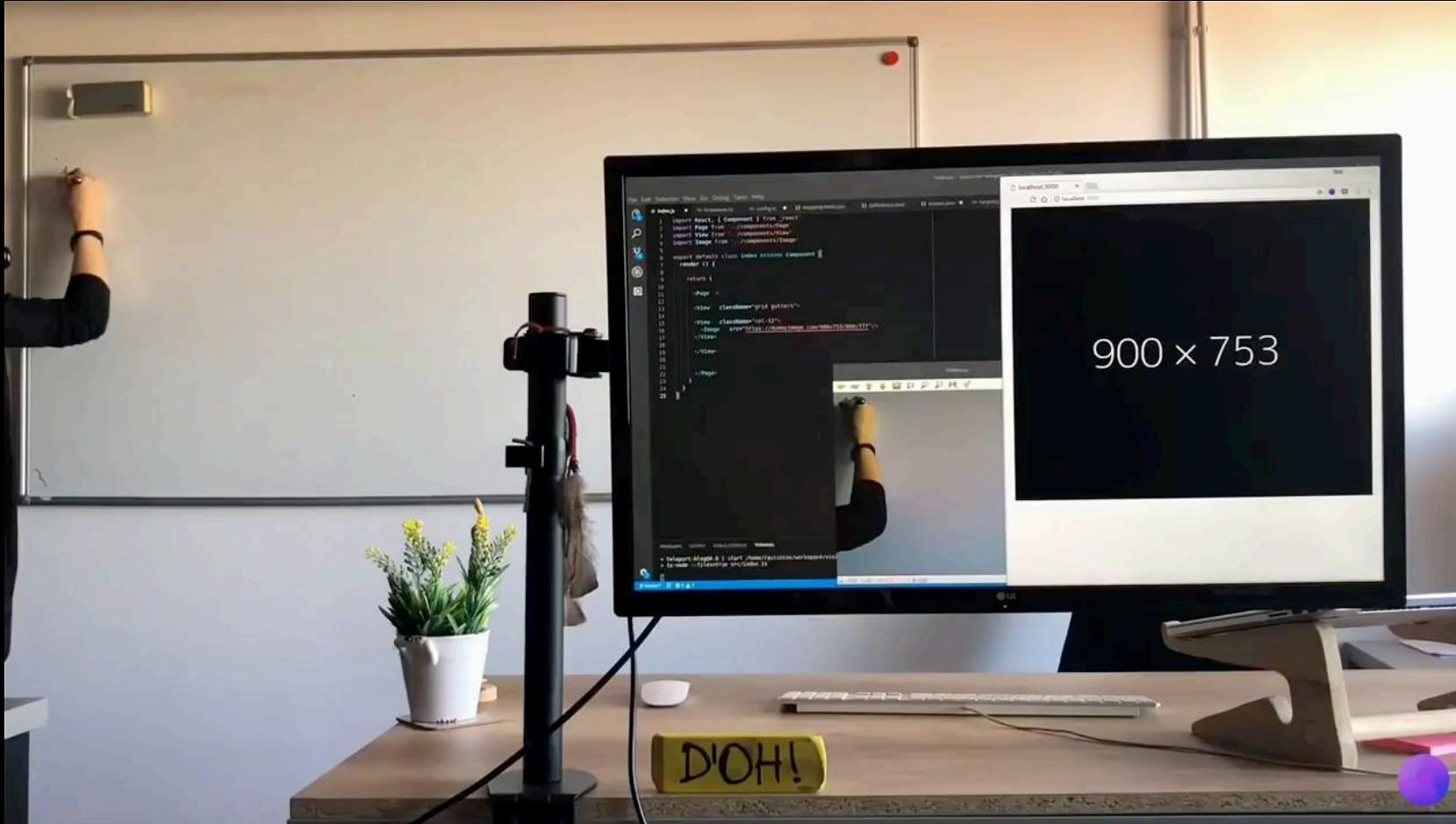
After Waste is removed in a kitchen (Sandwich making process example).

到底什么是研发效能？



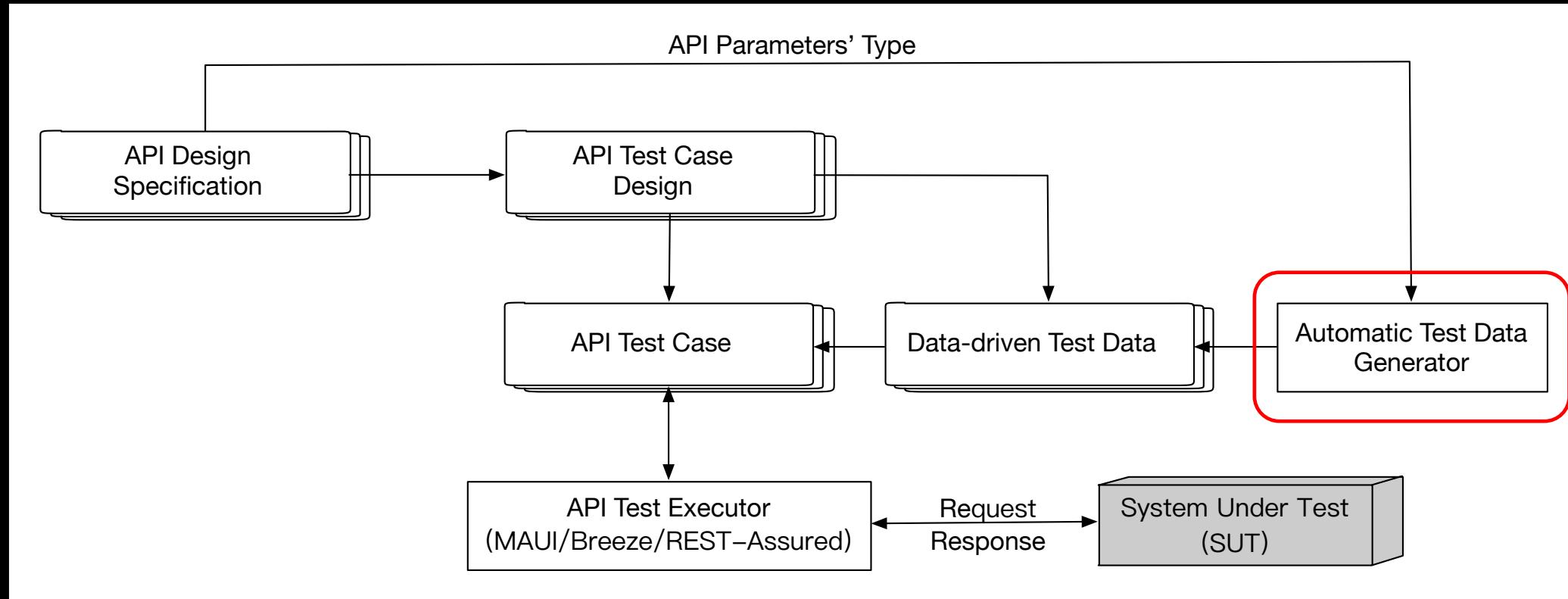
先让我们主观感受一下 研发效能提升之美

案例1：前端代码的自动生成



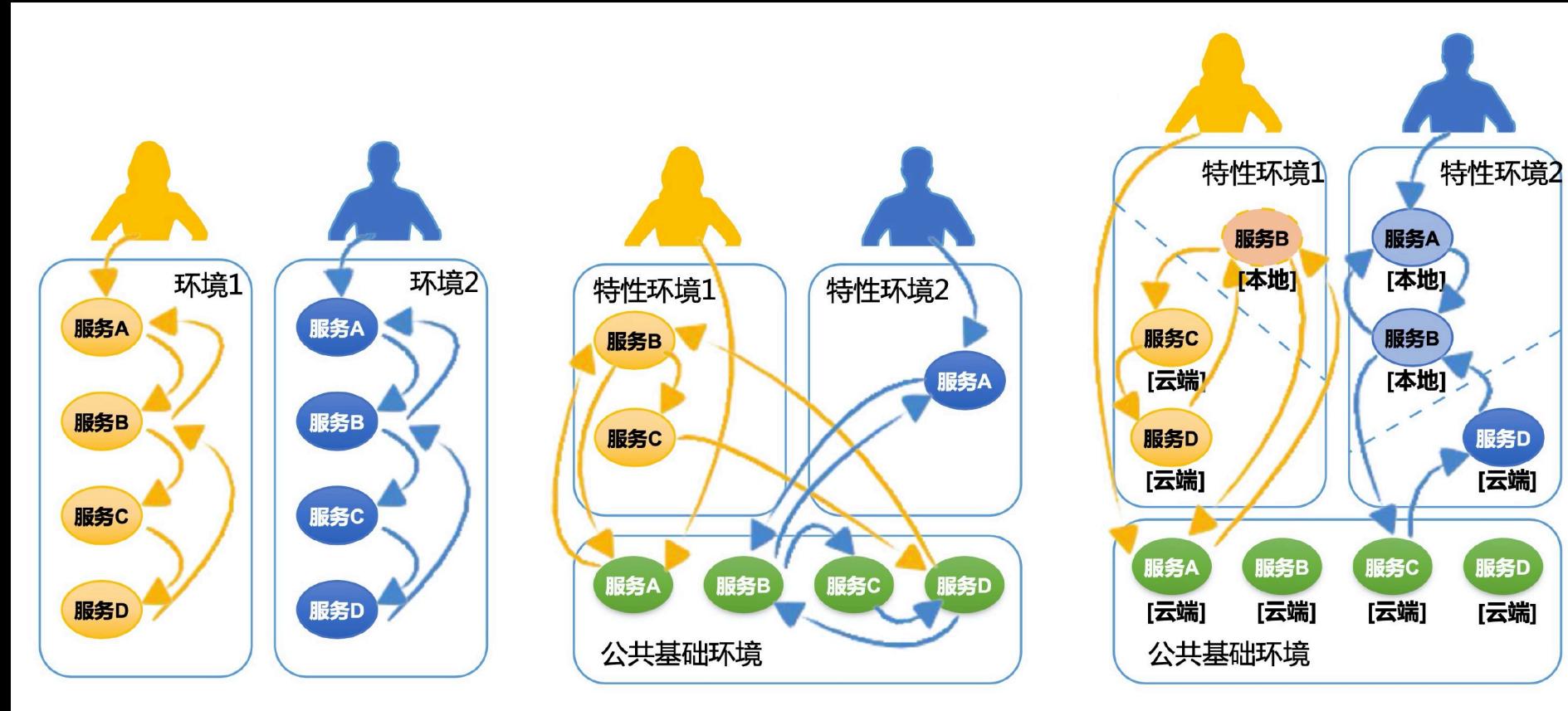
先让我们主观感受一下 研发效能提升之美

案例2：临界参数下的API测试



先让我们主观感受一下 研发效能提升之美

案例3：微服务架构下测试环境的困局



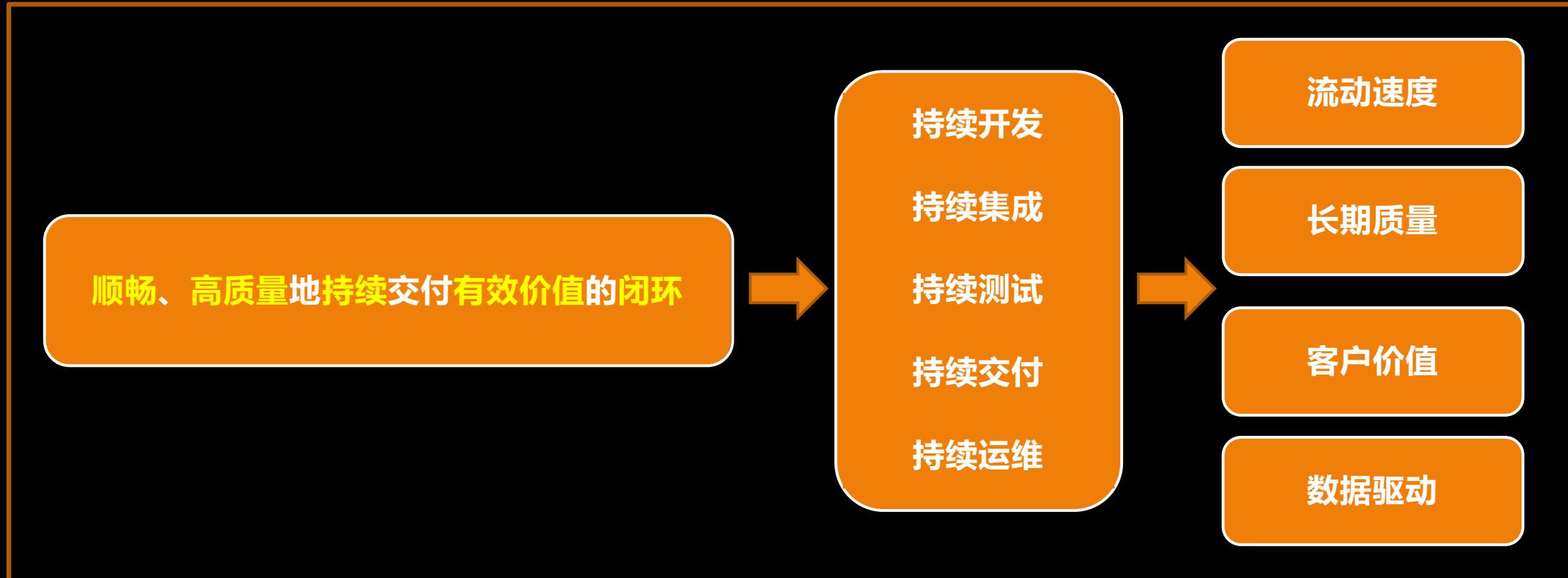
版本同步

链路巡检

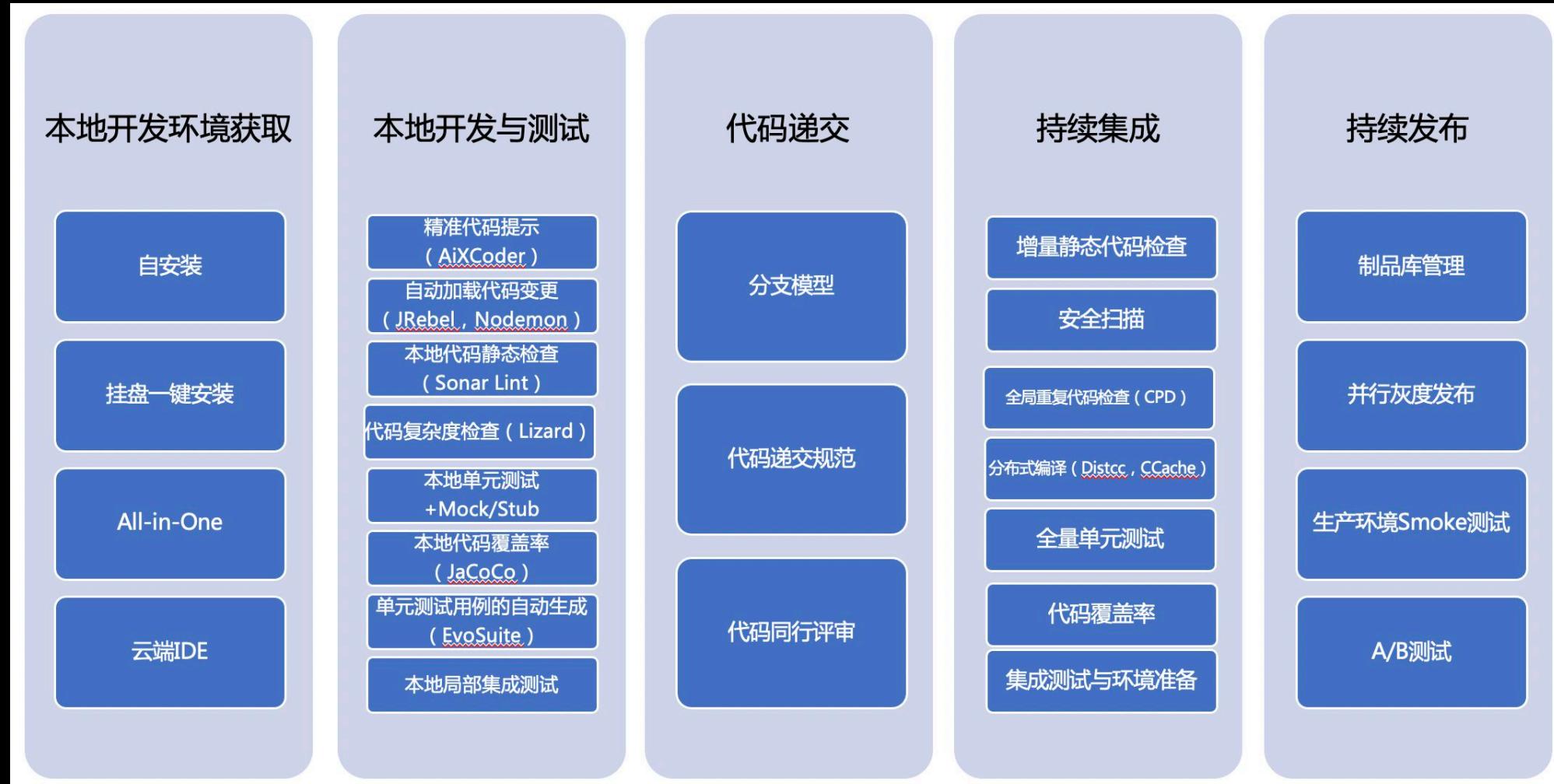
依赖管理

架构拓扑

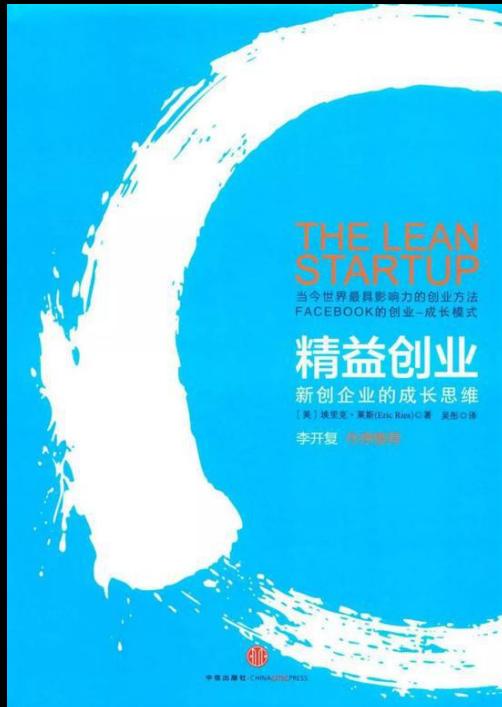
研发效能的“第一性原理”



“研发效能”的点点滴滴



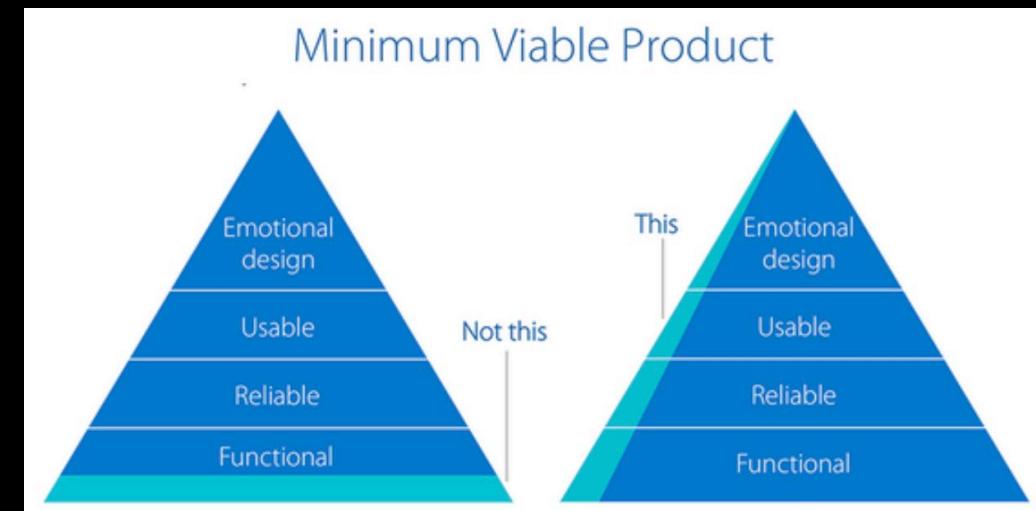
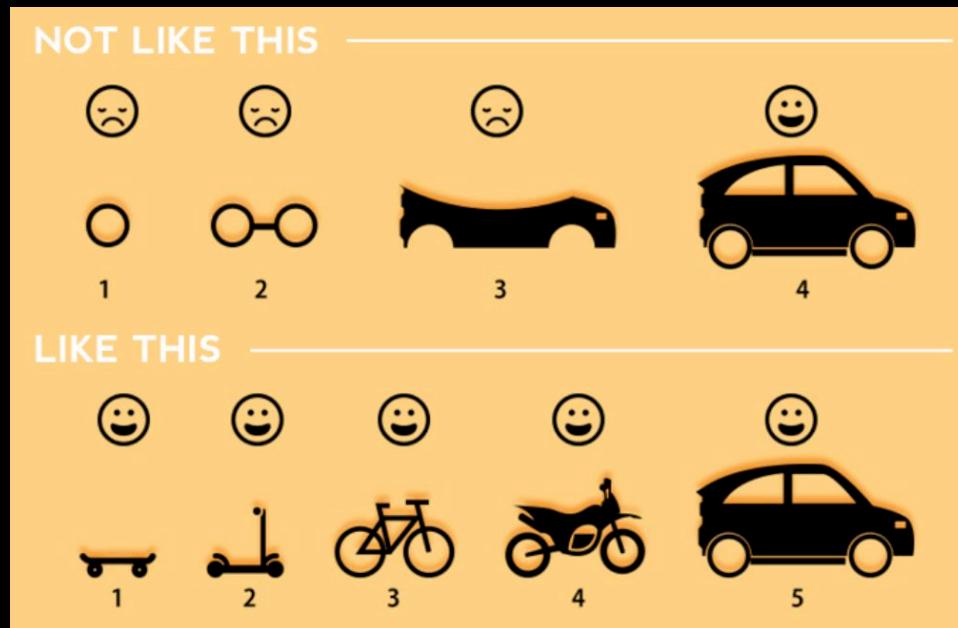
研发效能提升的一些经验和实践



用MVP (Minimum Viable Product) 的
思想来提升研发效能

研发效能提升的一些经验和实践

MVP方法的常见误区



研发效能提升的一些经验和实践

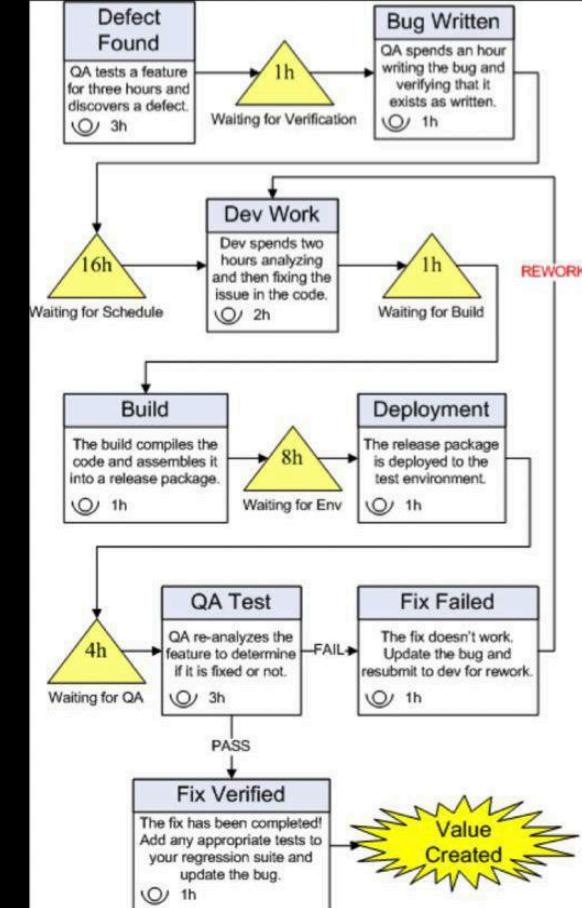
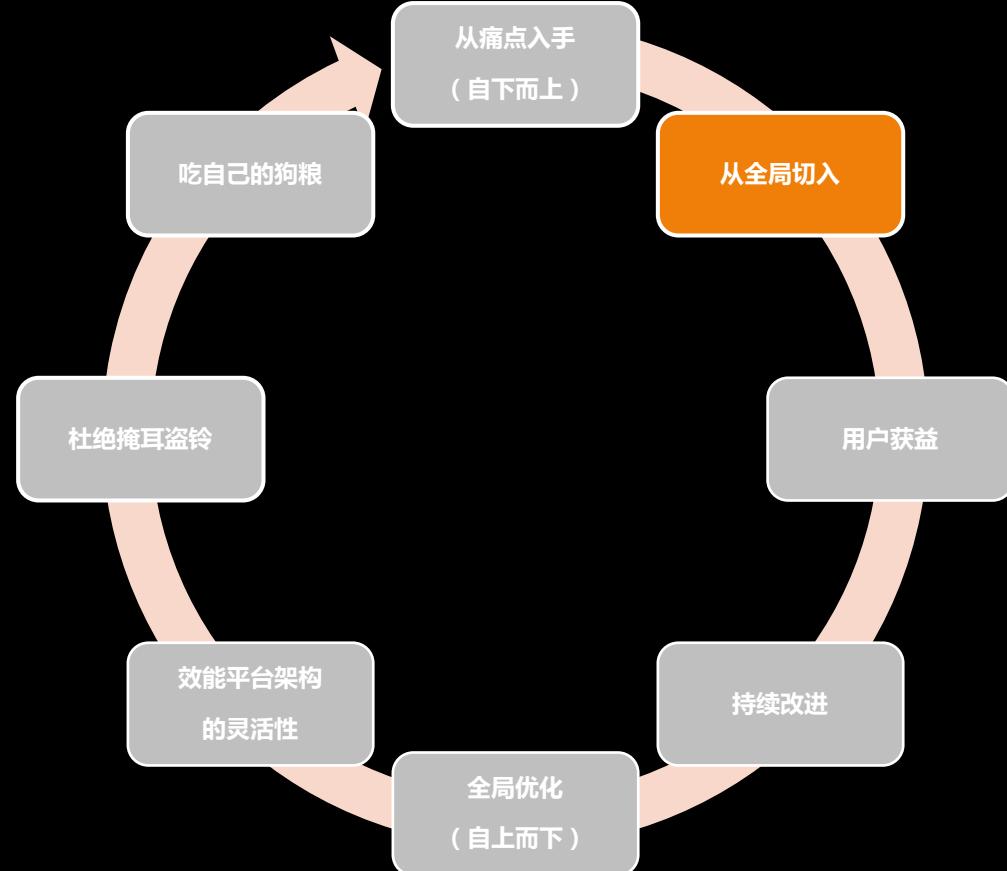


研发效能提升的一些经验和实践

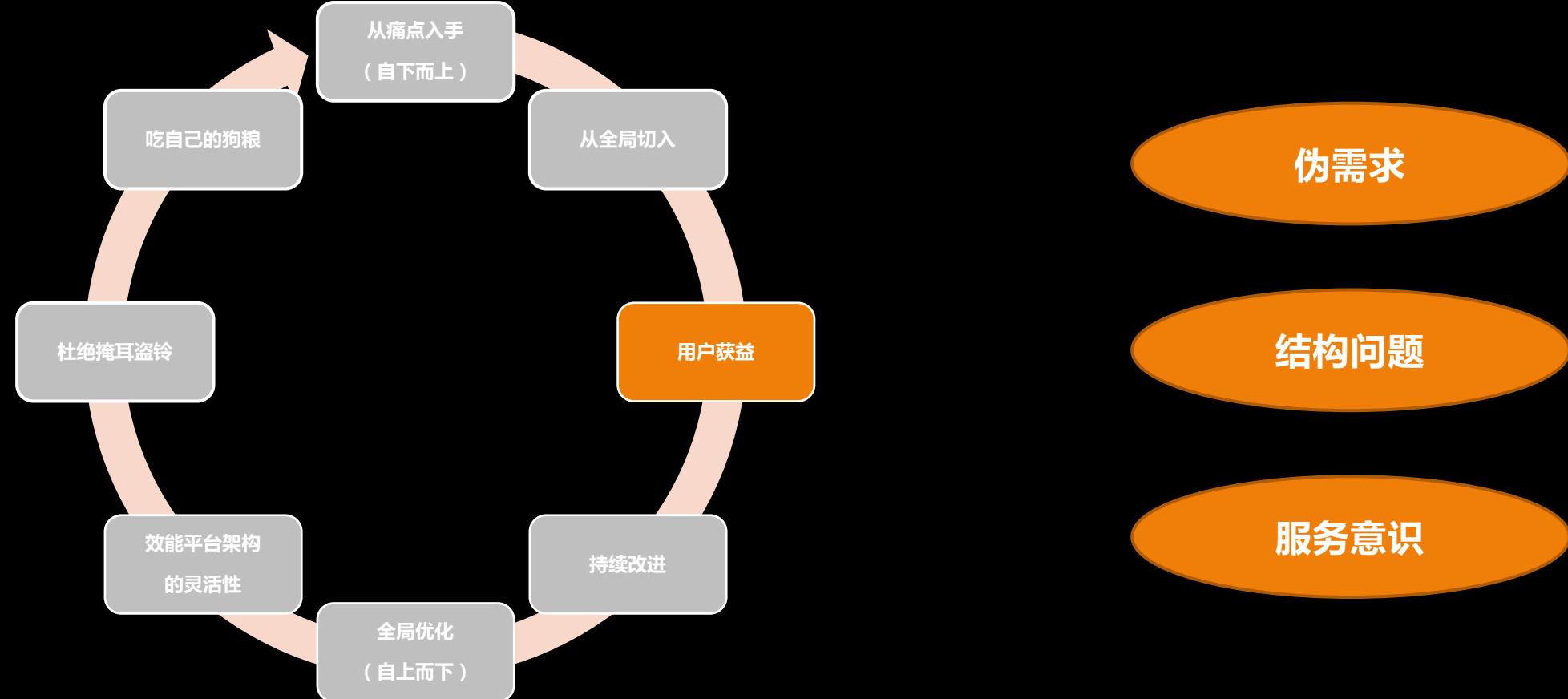


- 本地编译耗时长
- 本地测试困难，测试环境准备复杂且耗时
- 自动化测试用例数量大，执行和维护成本高
- 测试数据准备困难
- 研发后期阶段，代码递交集中，缺陷井喷
- 性能缺陷在研发后期发现，修复重测成本高居不下
- API频繁变更引发前后端集成阶段问题频现
- 集群规模庞大，发布过程耗时过长
- 项目的过程数据都是后期集中填充，失去度量意义
- ...

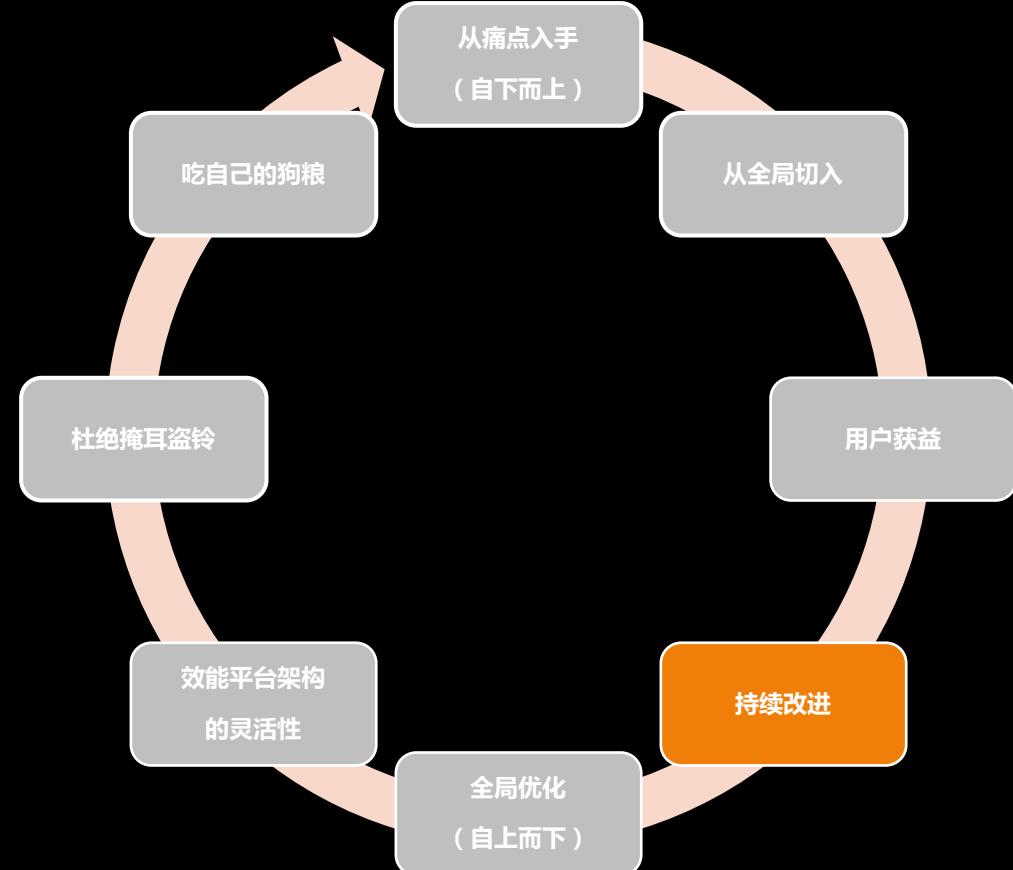
研发效能提升的一些经验和实践



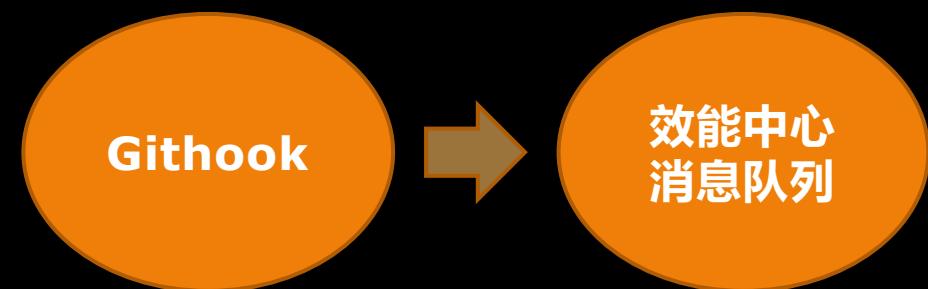
研发效能提升的一些经验和实践



研发效能提升的一些经验和实践

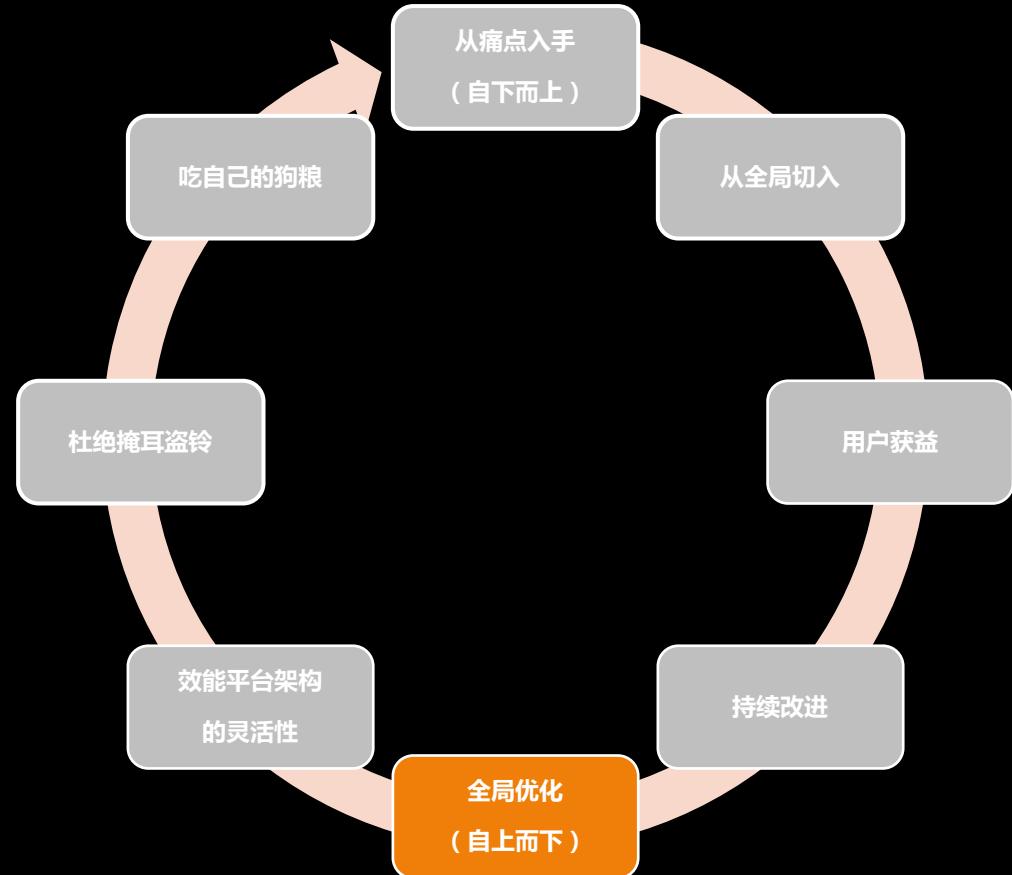


案例：Githook的机制真的是最佳的方案吗？

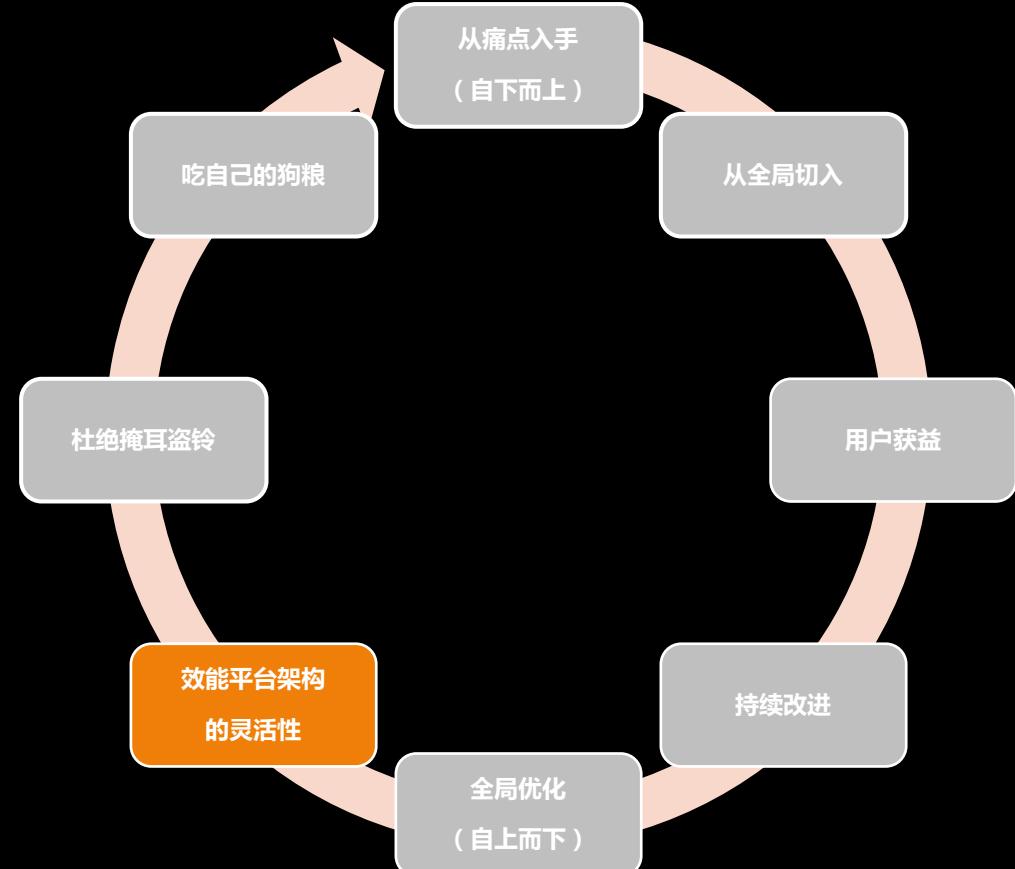


研发效能提升的一些经验和实践

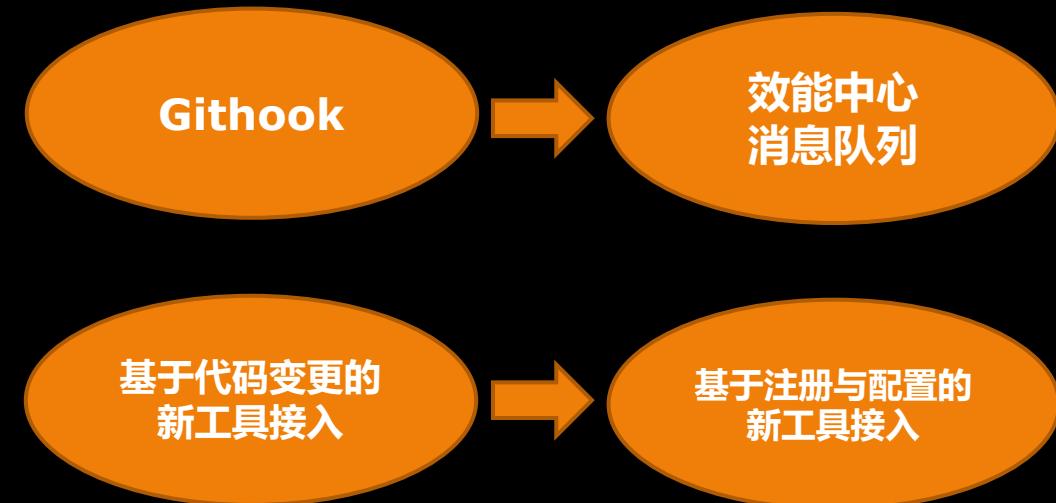
研发效能提升模式的选择



研发效能提升的一些经验和实践

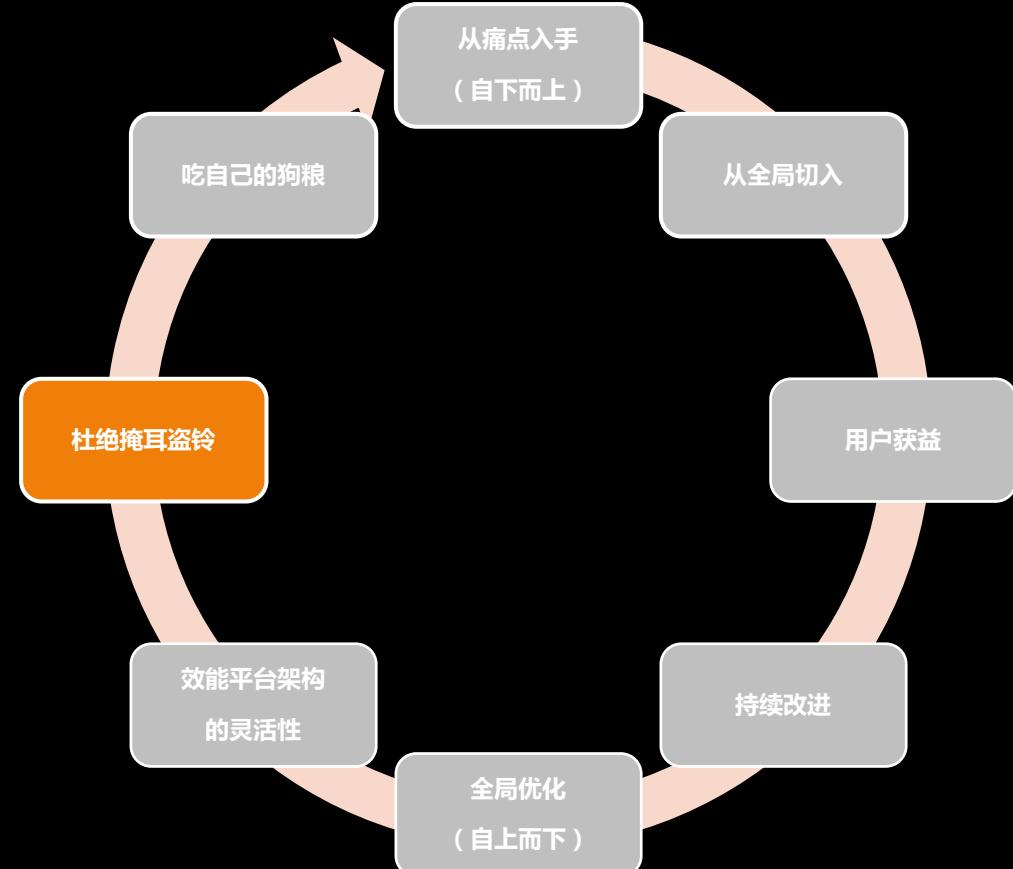


唱戏的 VS 搭台的



先搭好台，还是先唱好戏？

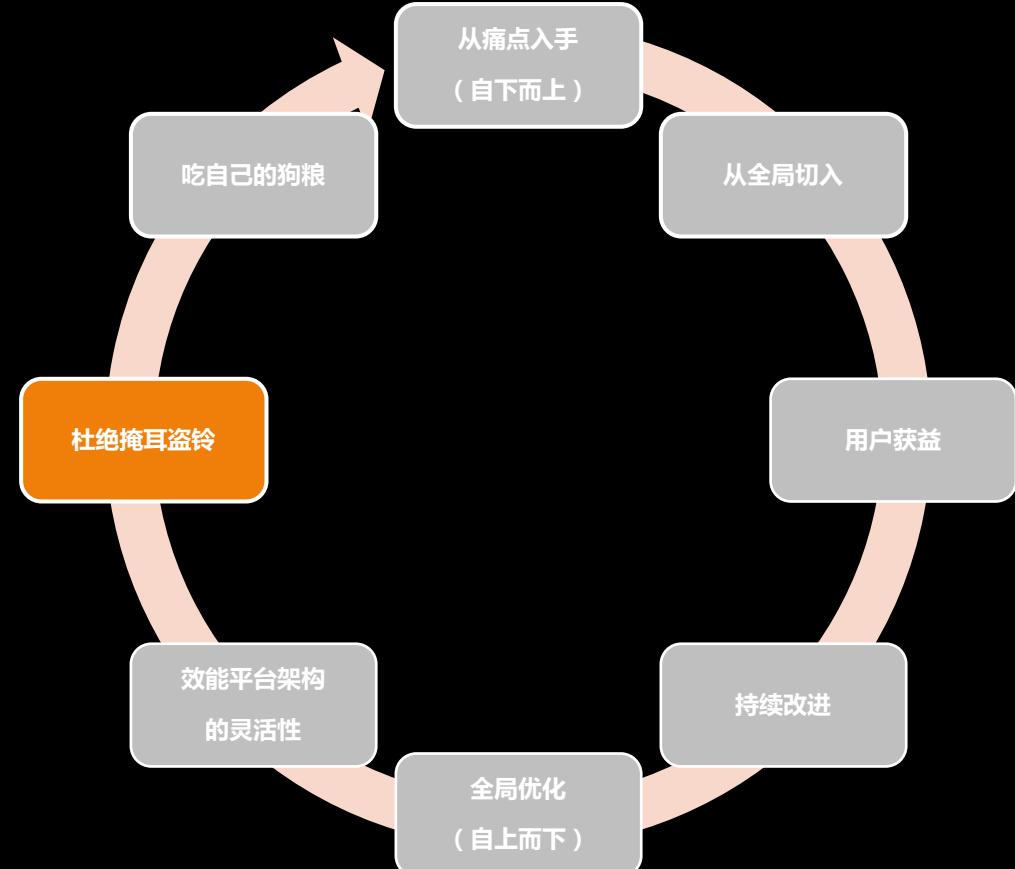
研发效能提升的一些经验和实践



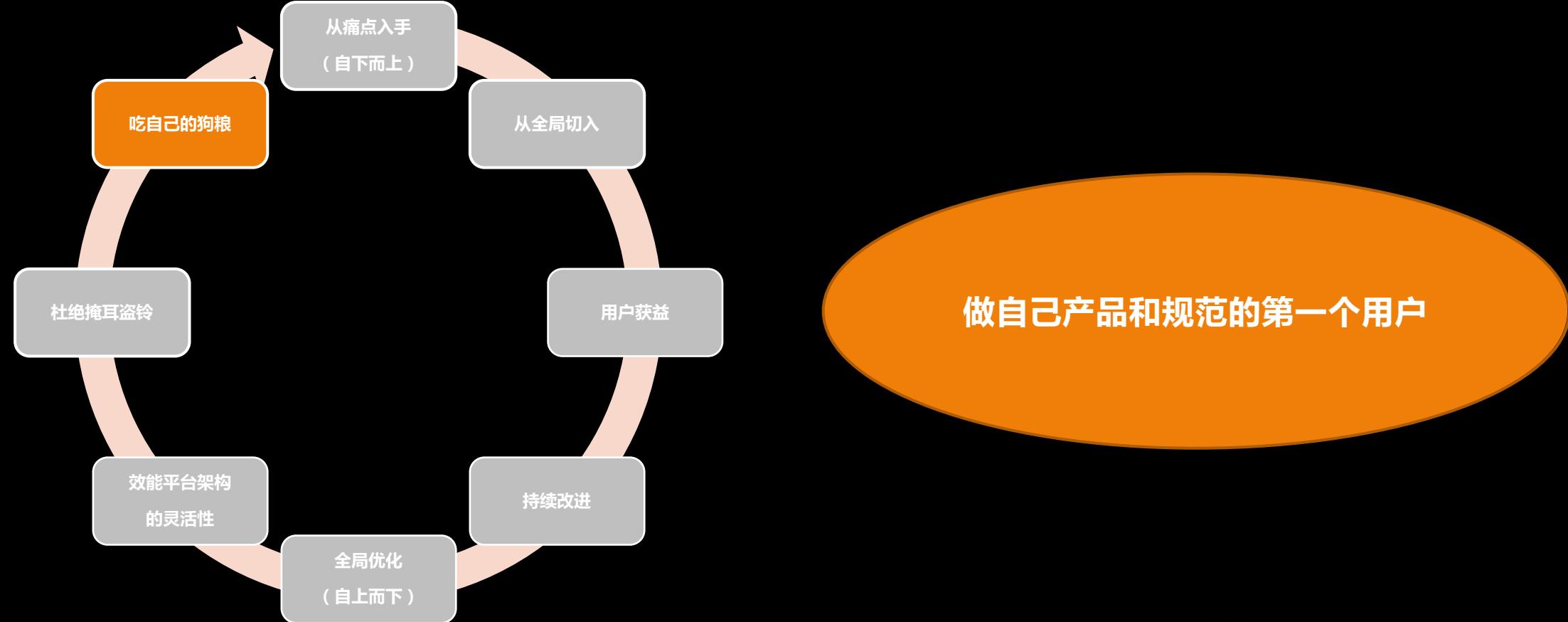
研发效能的“最差实践”

- 代码质量门禁Sonar设而不卡
- 单元测试只是执行，不写断言Assert
- 代码覆盖率形同虚设
- Peer Review走过场
- 代码递交随意递交
- 监控超配，有报警但无人认领
- ...

研发效能提升的一些经验和实践



研发效能提升的一些经验和实践



研发效能提升的行业趋势解读

研发流程全链路打通与过程可视化

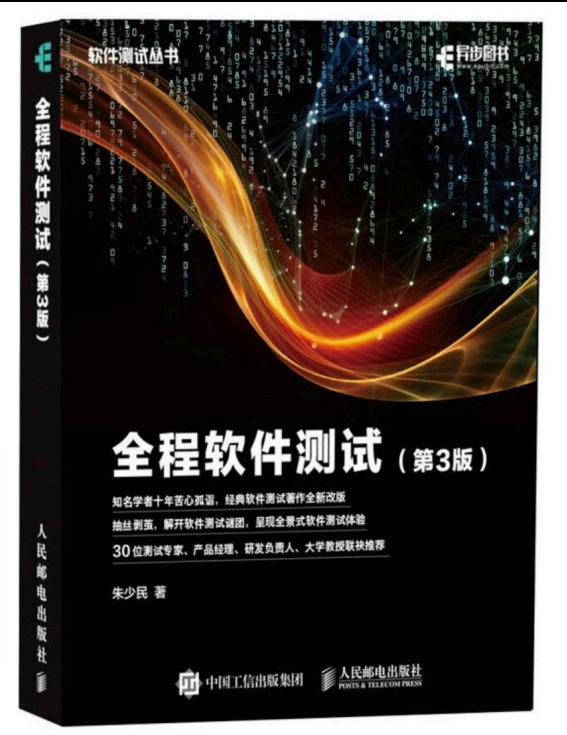
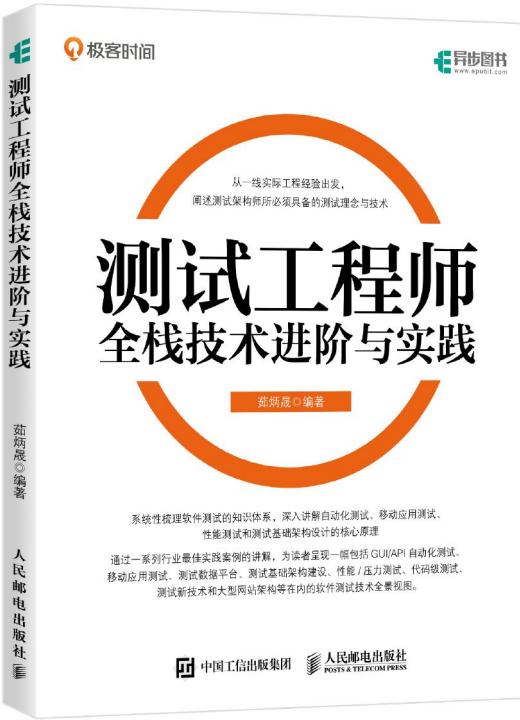
“敏态”和
“稳态”齐头并进

研发能力中台的沉淀

数据驱动下的
效能提升

研发效能“从
有到无”

推荐阅读



宋铜铃

华为某软件首席架构师



现任华为云 + AI BG 某软件首席架构师，有近 20 年的大规模嵌入式系统软件、应用软件的编码和设计经验，曾参与 ATCA 电信计算平台、服务器、Atlas、鲲鹏计算、公有云服务器运维管理软件的设计、编码和重构。

主办方：

Boolan
高端 IT 咨询与教育平台

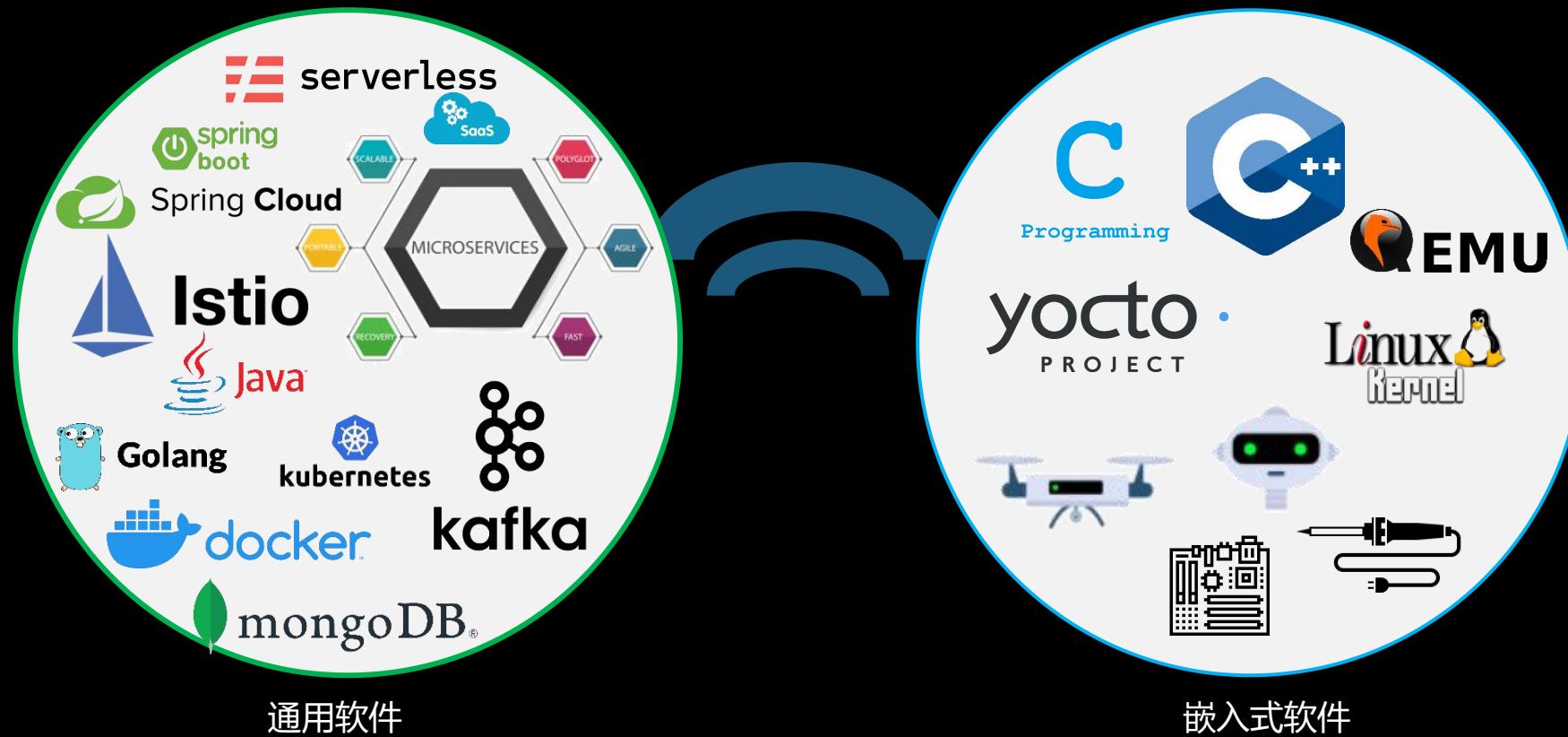
C++ Summit 2020

宋铜铃

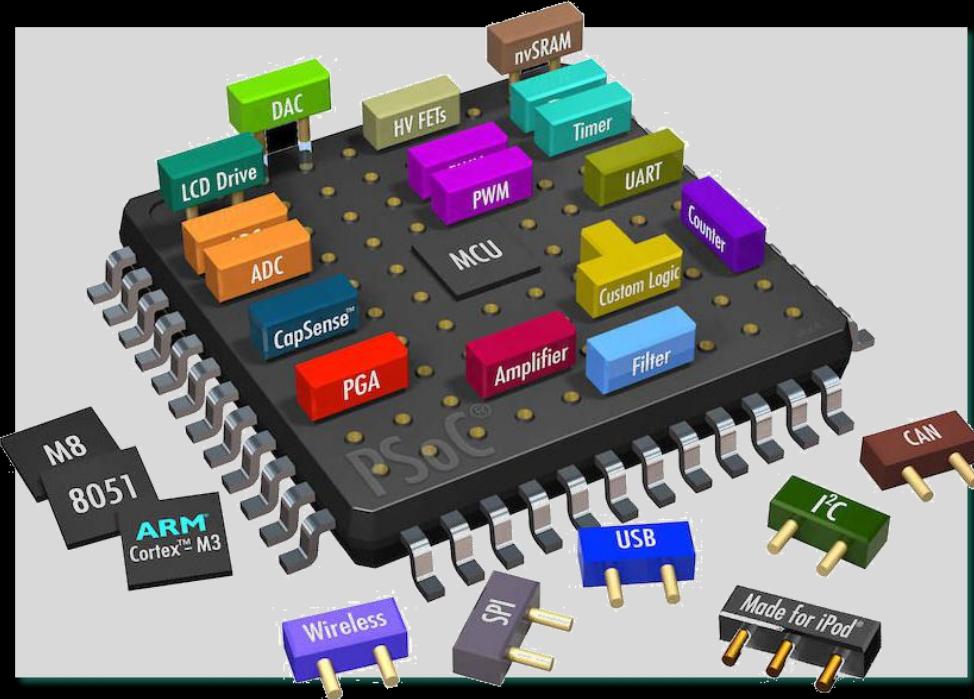
华为资深软件架构师

基于C语言的 微组件架构 实践

软件空间的两个平行世界



嵌入式系统开发的那些限制



CPU: ARM/MIPS/RISC/单片机

RAM: ~ MB/G

FLASH: ~MB/G

- 有限的硬件资源
- 以C语言为主要工作语言（团队技能）
- 缺少丰富的中间件、框架
- 不得不自己造轮子

C语言仍是主流

世界编程语言排行榜 TIOBE Index for November 2020

| Nov 2020 | Nov 2019 | Programming Language | Ratings | Change |
|----------|----------|----------------------|---------|--------|
| 1 | 2 | C | 16.21% | +0.17% |
| 2 | 3 | Python | 12.12% | +2.27% |
| 3 | 1 | Java | 11.68% | -4.57% |
| 4 | 4 | C++ | 7.60% | +1.99% |
| 5 | 5 | C# | 4.67% | +0.36% |
| 6 | 6 | Visual Basic | 4.01% | -0.22% |
| 7 | 7 | JavaScript | 2.03% | +0.10% |
| 8 | 8 | PHP | 1.79% | +0.07% |
| 9 | 16 | R | 1.64% | +0.66% |
| 10 | 9 | SQL | 1.54% | -0.15% |

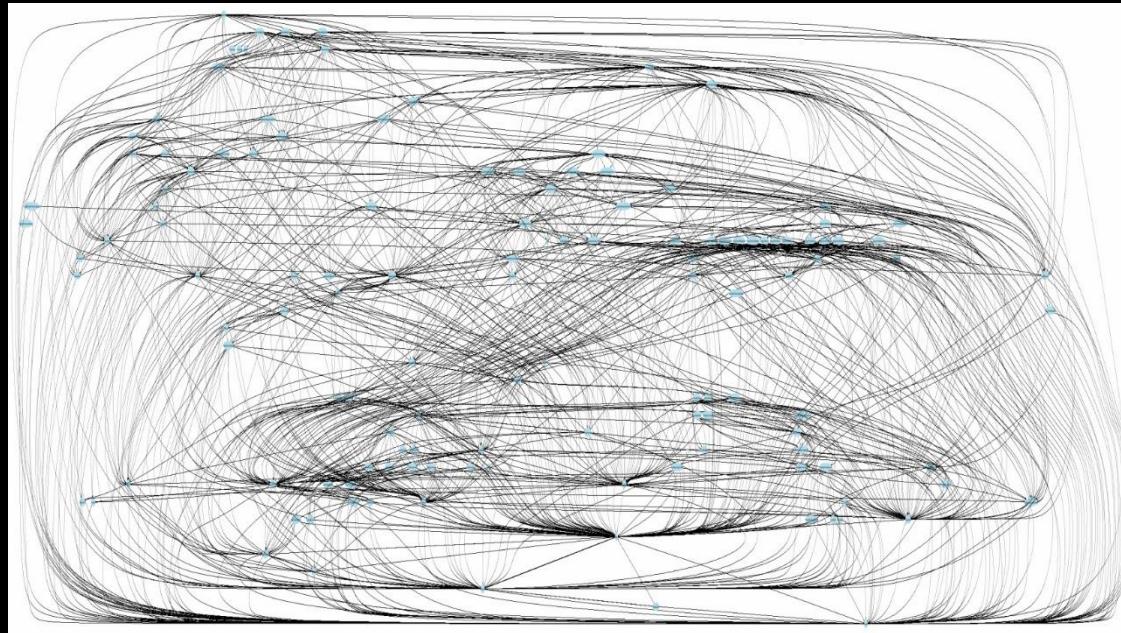
C程序的庞然大物让人吃惊

—《如何看待华为1100亿行代码.....》



C程序更易失控

缺少成熟的平台框架、不支持原生OOP、脆弱的内存安全性



泥团架构噩梦

- 可读性急剧下降，不敢动代码
- Bug漫天飞舞
- 无数测试用例，人海战术

面向变化的脆弱性

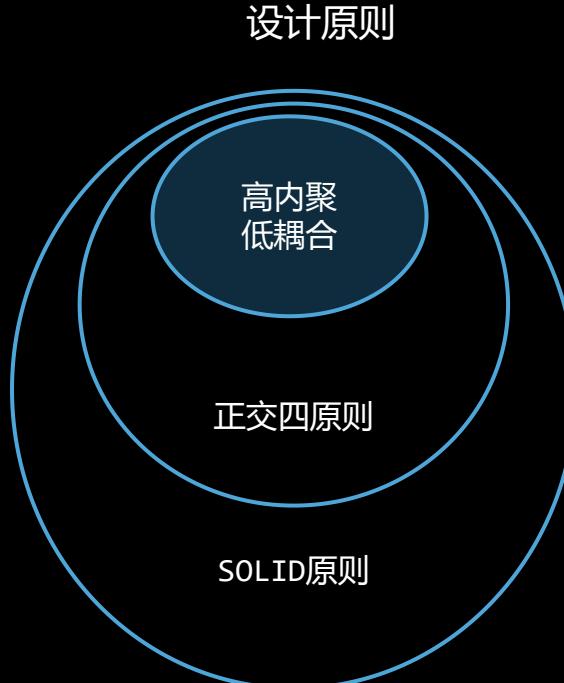
- 风吹草动，改代码

似曾相识的问题

- 修改一行代码，要理解1000行代码，网状的依赖
- 腐化扩散，脆弱的接口隔离
- 对外的接口、内部接口、业务代码没有明显的边界
- 靠规章制度，而不是技术手段来开展接口的看护
- 添加功能困难，去掉不需要的功能也困难
- MR相互等待，合代码效率低，解决不完的git冲突
- 模块无法单独构建，必须要编译整个工程
- 模块无法单独调试，测试，需要把整个系统跑起来，需要一群人开展调试
- 只有把软件加载到真实的硬件上才能运行调试
-

--架构有问题

两个世界，大道归一



设计模式

- 抽象工厂 (abstract factory)
- 建造者
- 单例模式
- 适配器(Adapter)
- 桥接(Bridge)
- 外观(Facade)
- 迭代器(Iterator)
- 观察者(Observer)
- 模板方法(Template Method)
-

架构属性

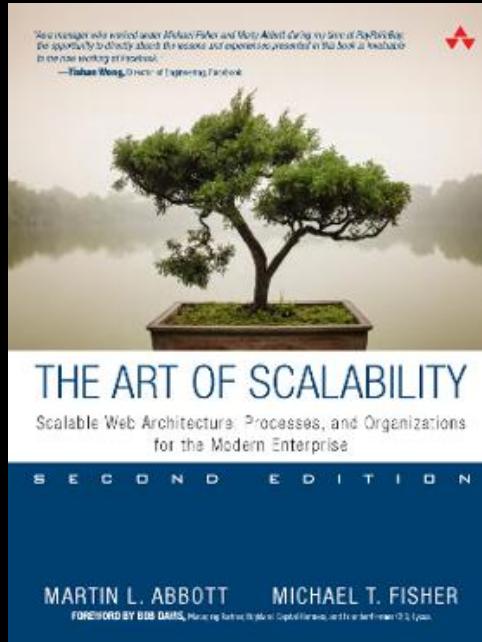
- 易用性
- 可靠性
- 性能
- 可扩展性
- 可构建性
- 可重用性
- 可部署性
- 可裁剪性
-

架构方法

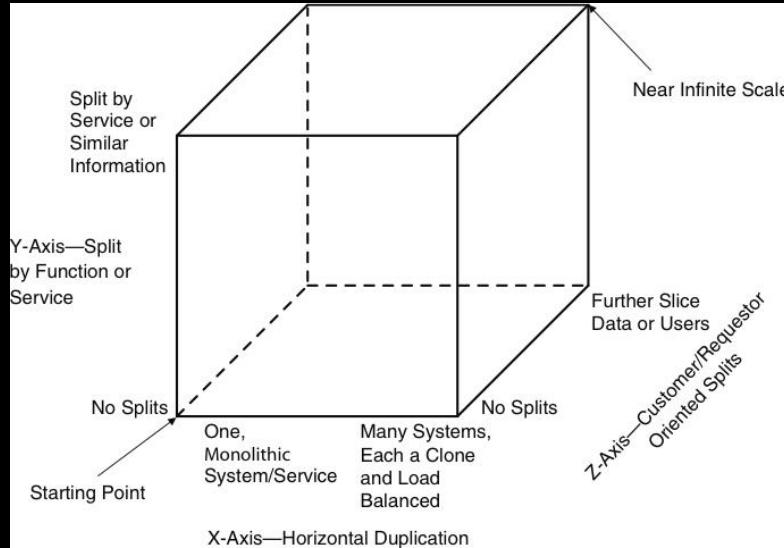
- DDD
- TDD
- BDD
-

所有的原则和方法是为了做好两件事：**拆分(划边界)、定接口**

拆分立方，《The art of scalability》



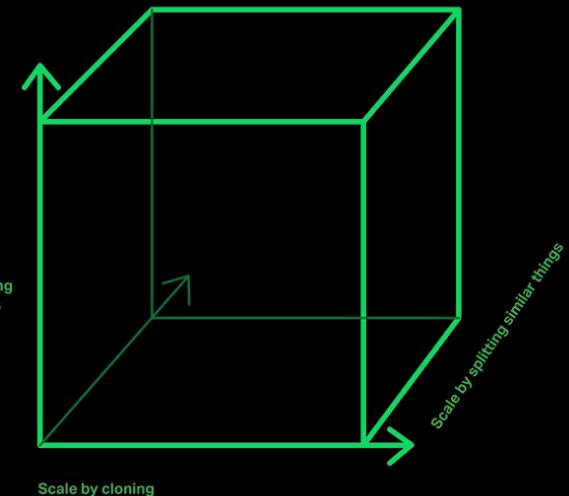
迈克尔·T·费希尔 和马丁·L·艾勃特 2009



X轴：单实例->多实例拆分

Y轴：功能拆分

Z轴：数据分区



微服务扩展立方

拆分即架构

拆分的目的是为了隔离不同频的变化

为谁拆分

为了性能而拆分.....

为了可靠性而拆分.....

为了解耦而拆分.....

为了可替换/可裁剪而拆分.....

为了可构建而拆分.....

怎么拆分

拆分进程、线程.....

拆分模块.....

拆分文件.....

拆分数据.....

拆分代码仓.....

接口

进程间:

- Signal
- Pipe
- Socket
- RPC
- Restful
- Semaphore
- Shared Memory
- ...

进程内:

- 函数
- 变量

iBMC®软件的架构实践

- iBMC®是一个系统软件，包含OS、业务代码以及用户接口。以C语言为主，包含C/C++/Rust/javascript，自研代码规模将近200万行
- iBMC®先后支持超过100款华为产品，已广泛应用在全球主流电信运营商、国内各大一二三线互联网公司
- 在多款产品中，iBMC®保持了一套主干代码以及组件级的二进制共享



架构挑战：

- 1、需要运行在多种嵌入式SoC上，CPU指令集可能存在差异，芯片驱动各不相同
- 2、需要支持数千种硬件；
- 3、用户需求持续活跃；
- 4、需要支持多团队异步并行开发；

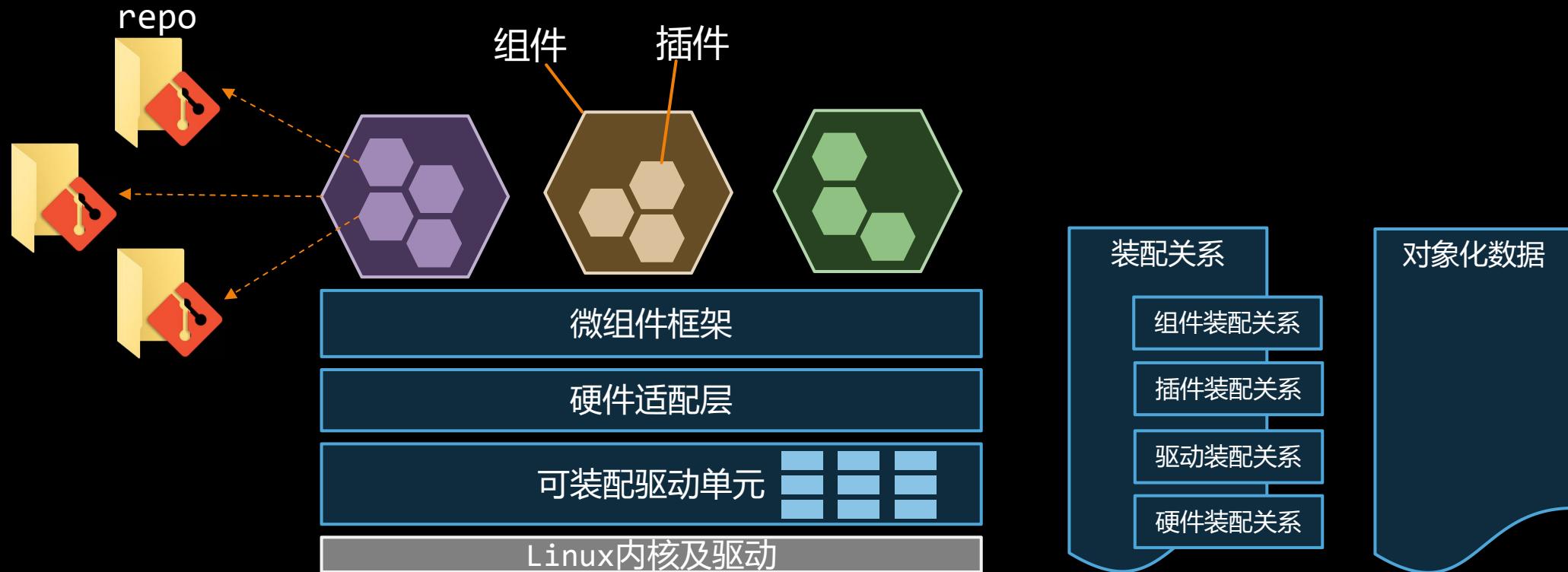
解决架构挑战→[微组件架构实践](#)

微组件架构的12个特征

- 粒度可小于进程
- 可独立开发
- 可独立构建
- 可独立调试
- 可独立测试
- 可裁剪
- 可替换
- 接口支持OOP
- 接口延时绑定
- DAG依赖
- 接口可视（支持静态检查、支持工具辅助开发）
- 依赖关系可视（支持静态检查、支持工具可视化）

微组件架构

组件以进程为粒度，插件以函数为粒度。插件可以轻量至100行代码



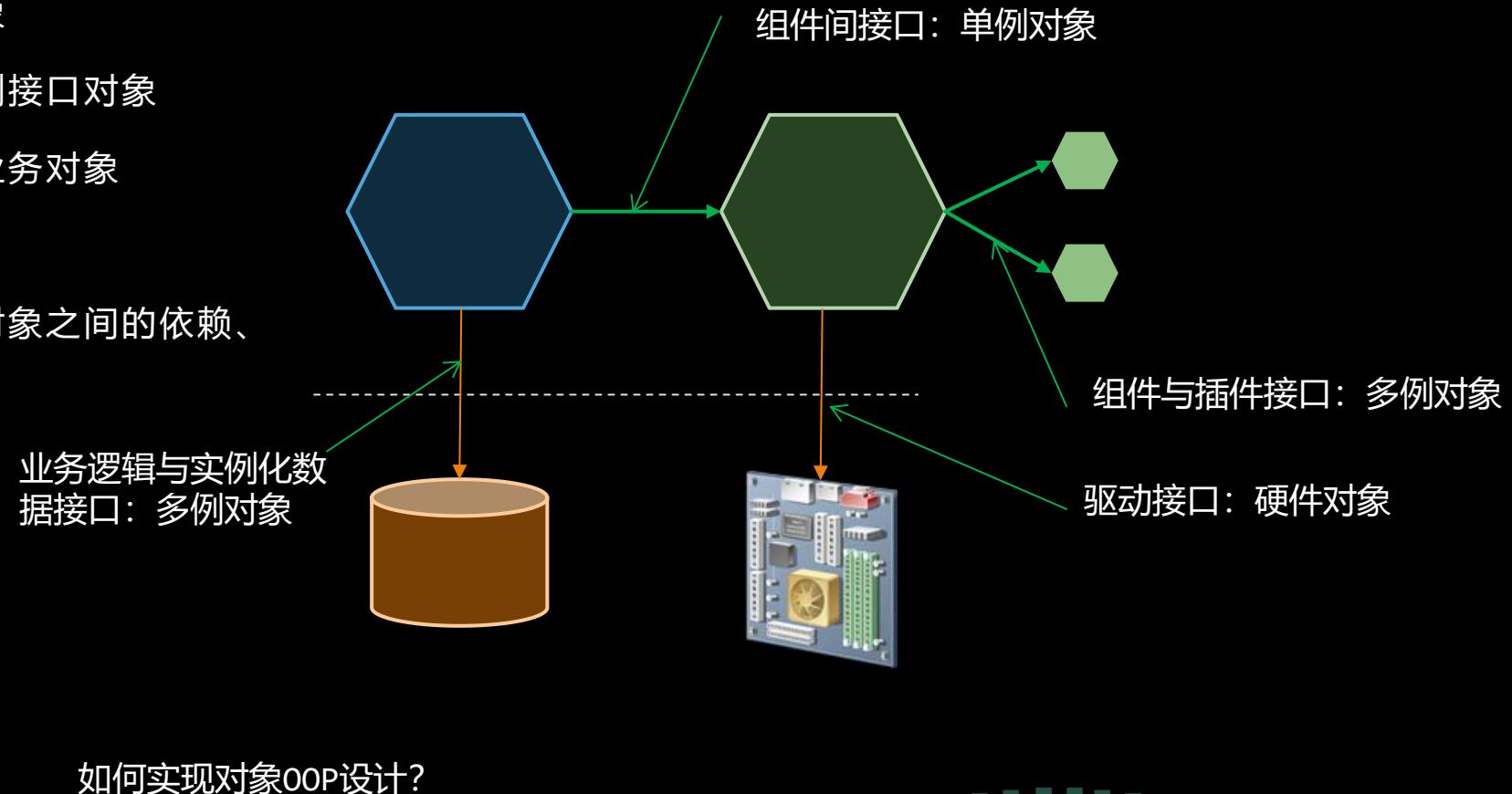
基于微组件架构的拆分

拆分不同频的变化：

- 不同功能域→拆分为不同组件
- 同一功能域，通用的和特定的行为→拆分为组件和插件
- 行为和数据→拆分为代码和类（对象）
- 驱动按照稳定要素拆分：
 - 芯片是稳定的，芯片的组合、硬件的组合是可变的→拆分为芯片驱动与拓扑驱动，
 - 芯片驱动拆分：芯片是稳定的，但其地址、状态是可变的→拆分为芯片驱动类（代码）+芯片对象实例（数据）

一切皆对象，对象即接口

- 组件间的接口：单例接口对象
- 组件与插件间的接口：多实例接口对象
- 代码中的抽象模型：多实例业务对象
- 驱动：硬件对象
- 依赖关系：类之间的依赖、对象之间的依赖、组件/插件间的依赖



如何实现对象OOP设计？

C++ OOP回顾

```

class Processor {
protected:
    string vendor;
    int numberOfCores;
    double frequency;

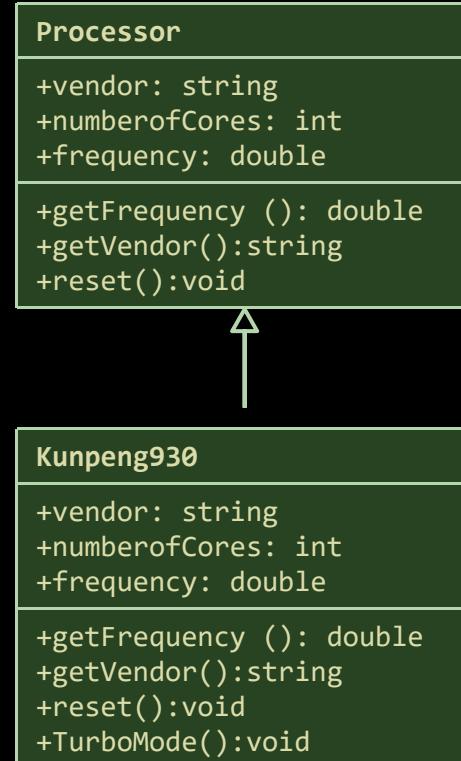
public:
    virtual void reset() const = 0;
    virtual double getFrequency() const { return frequency; };
    virtual string getVendor() const { return vendor; };
};

class Kunpeng930 : public Processor {
public:
    // virtual double getFrequency() const;      // 继承
    virtual string getVendor() const override; // 重写
    virtual void reset() const;                // 实现
    virtual void TurboMode();                  // 扩展
};

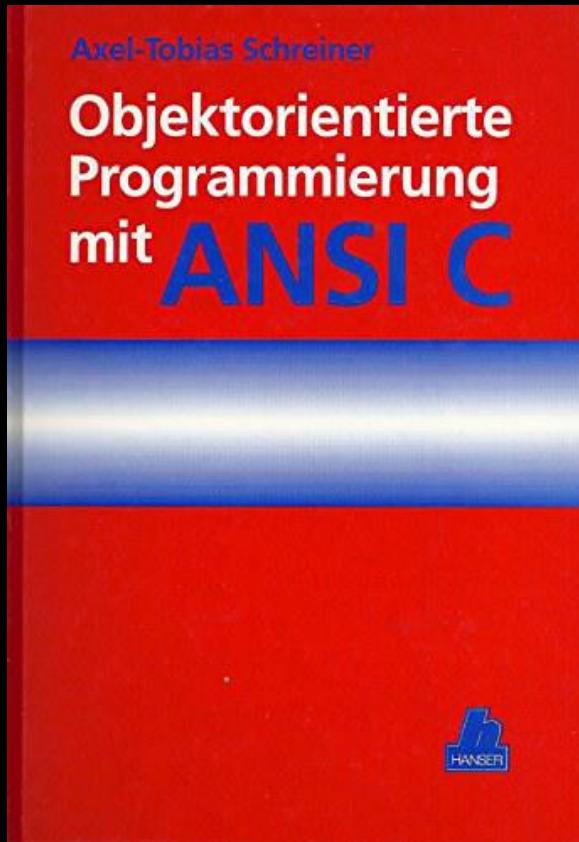
Processor *cpu = new Kunpeng930(string("Huawei"));
cout << cpu->getVendor() << endl; // 多态

```

UML



奇技yin巧：C的OOP实现——闲情逸致



作者: Axel-Tobias Schreiner

```
struct Class {
    void (* draw) (const void * self);
};

void draw (const void * self) {
    const struct Class * const * cp = self;
    (* cp) -> draw(self);
}

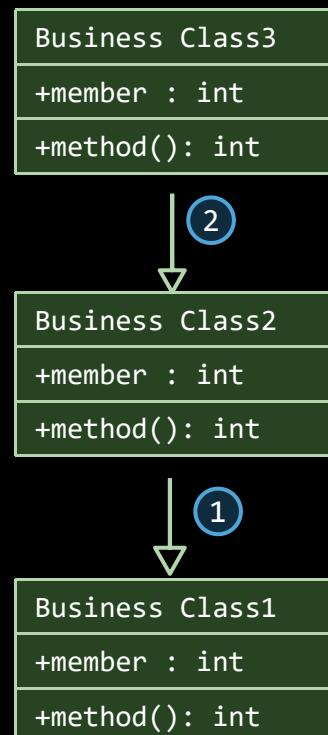
struct Point {
    const void * class;
    int x, y;
};
static void Point_draw (const void * _self) {
    const struct Point * self = _self;
    // draw ...
}

static const struct Class _Point = {Point_draw };
const void * Point = & _Point;
```

立体的 OOP——架构级 OOP

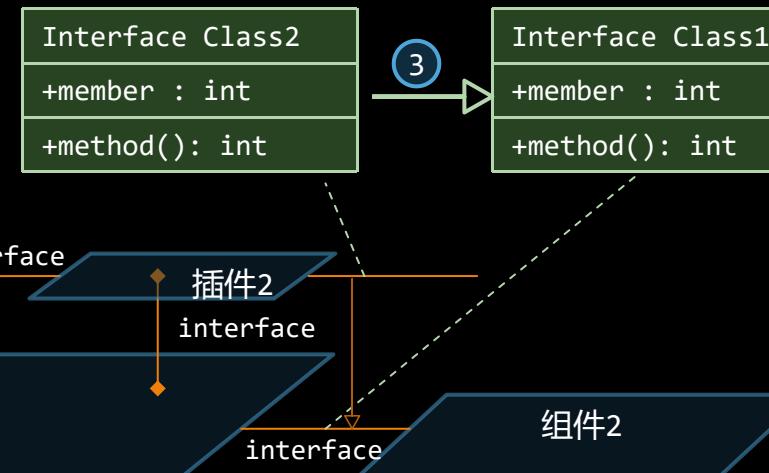
封装:

- 业务逻辑—业务类/对象
- 接口—接口类/对象



继承与多态:

- 组件内业务逻辑类的继承，组件与插件之间的业务逻辑类的继承
- 组件的接口的继承
- 组件与插件接口的继承



类用什么语言表达?

DSL & GPL

- DSL 语言与 GPL 语言的正确组合，各取所长：DSL—接口编程；GPL—业务逻辑
- DSL：面向承载系统架构设计目的而精心设计的胶水语言，极轻量语言解析框架
- 系统级OOP和语言级OOP。系统级OOP了，语言级OOP还重要吗
- 混合语言编程

DSL (interface, class, object)

```
namespace: Processor
interface : Cpu
methods:
int SetFrequency(double);
int SetTurboMode(int);
```

```
namespace: Processor
class : Processor
properties:
    string vendor;
methods:
    double GetVendor();
```

```
class : Kunpeng930 extends Processor
object : cpu1 from Kunpeng930
properties:
    string vendor="Huawei";
```

GPL (code)

```
int setFrequency(double
frequency)
{
    ...
}
int setTurboMode(int mode)
{
    ...
}
```

```
const char* getVendor(void)
{
    return strVendor;
}
```

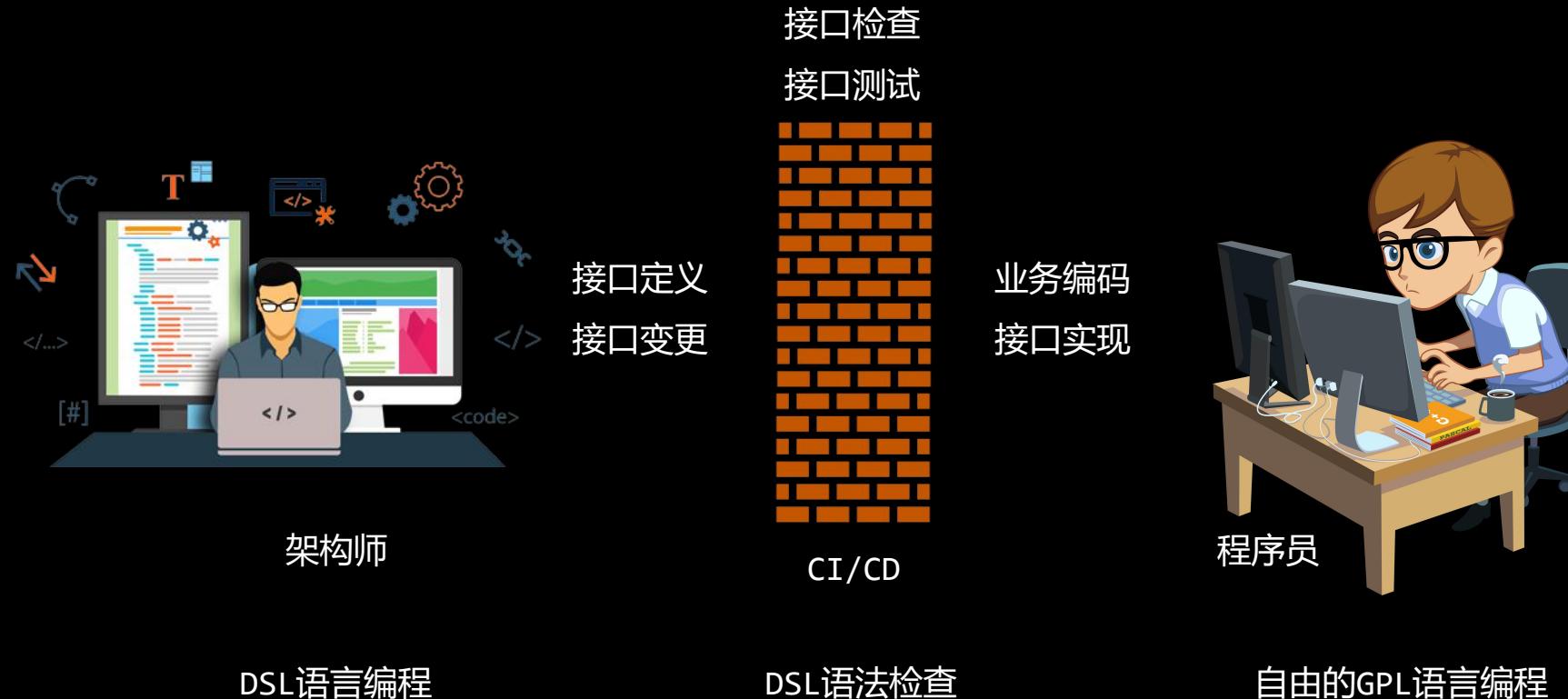
延迟绑定

- 运行时确定代码与对象的关系
 - 接口最终由谁实现？
- 对象在编码时不确定，在运行时构造
- 无编译依赖以实现独立的构建、调测
- 延迟绑定可支持组件/插件的动态替换或业务对象的动态变更

```
interface("Cpu") {  
    method("SetFrequency", setFrequency)  
    method("SetTurboMode", setTurboMode)  
}
```

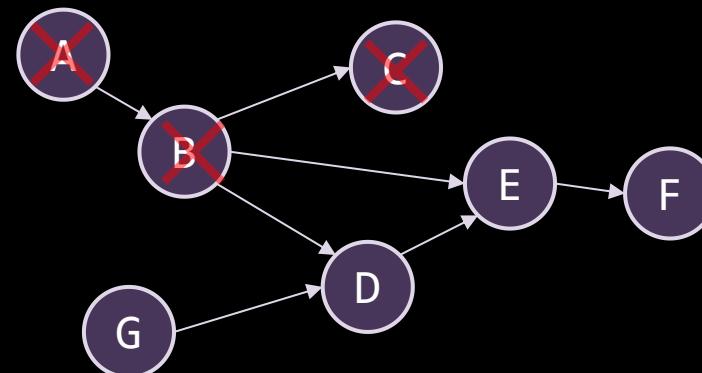
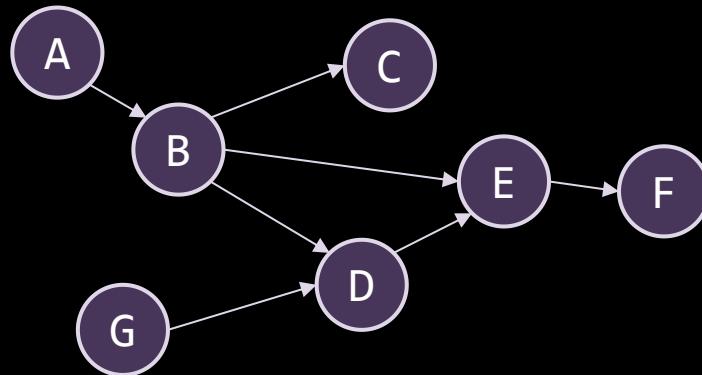
```
class("Kunpeng930") {  
    method("GetVendor", getVendor)  
}
```

“上帝的归上帝，凯撒的归凯撒”



DAG依赖

- 组件间的依赖
- 插件间的依赖
- 类之间的依赖
- 对象之间的依赖
- DAG依赖确保了系统构建时可裁剪

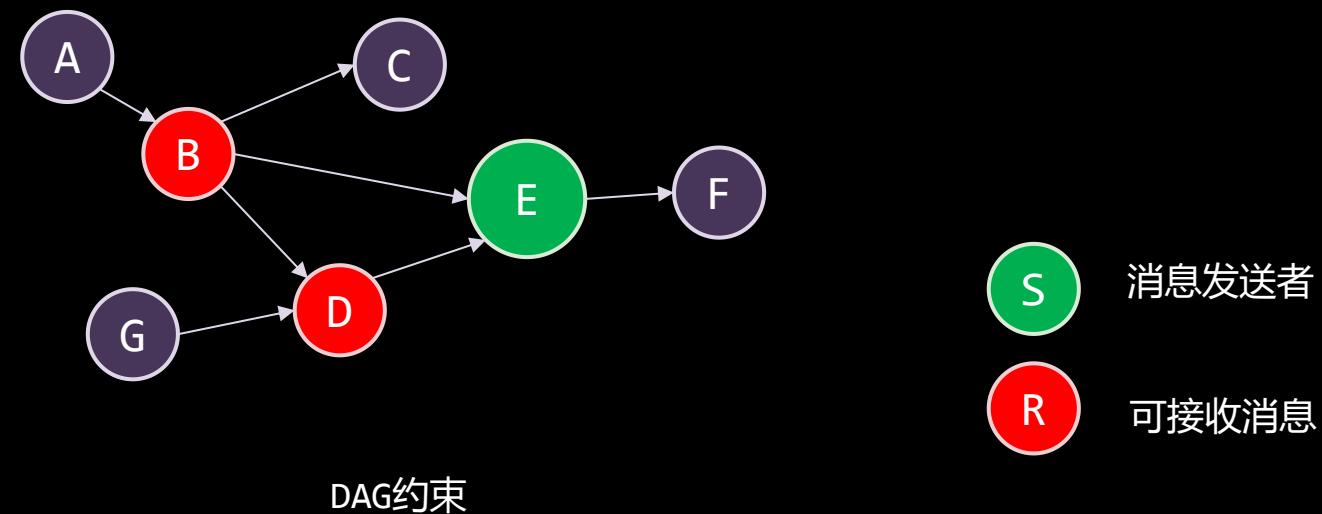


消息也是造成隐性依赖的重灾区

消息通讯：

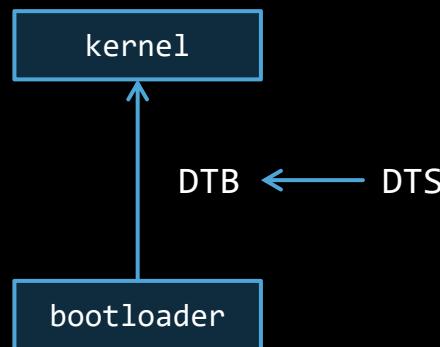
- zeromq
- rabbitmq
-

发送者阻止不了接收者的监听



驱动解耦——linux DTS

- Linux DTS(Device Tree Source)设备树

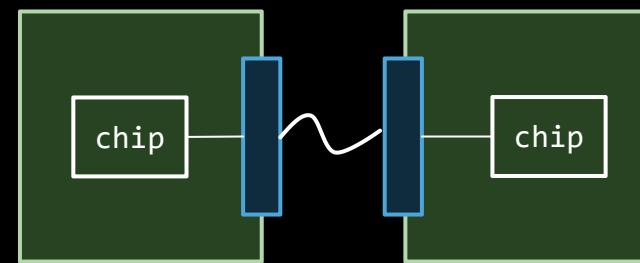
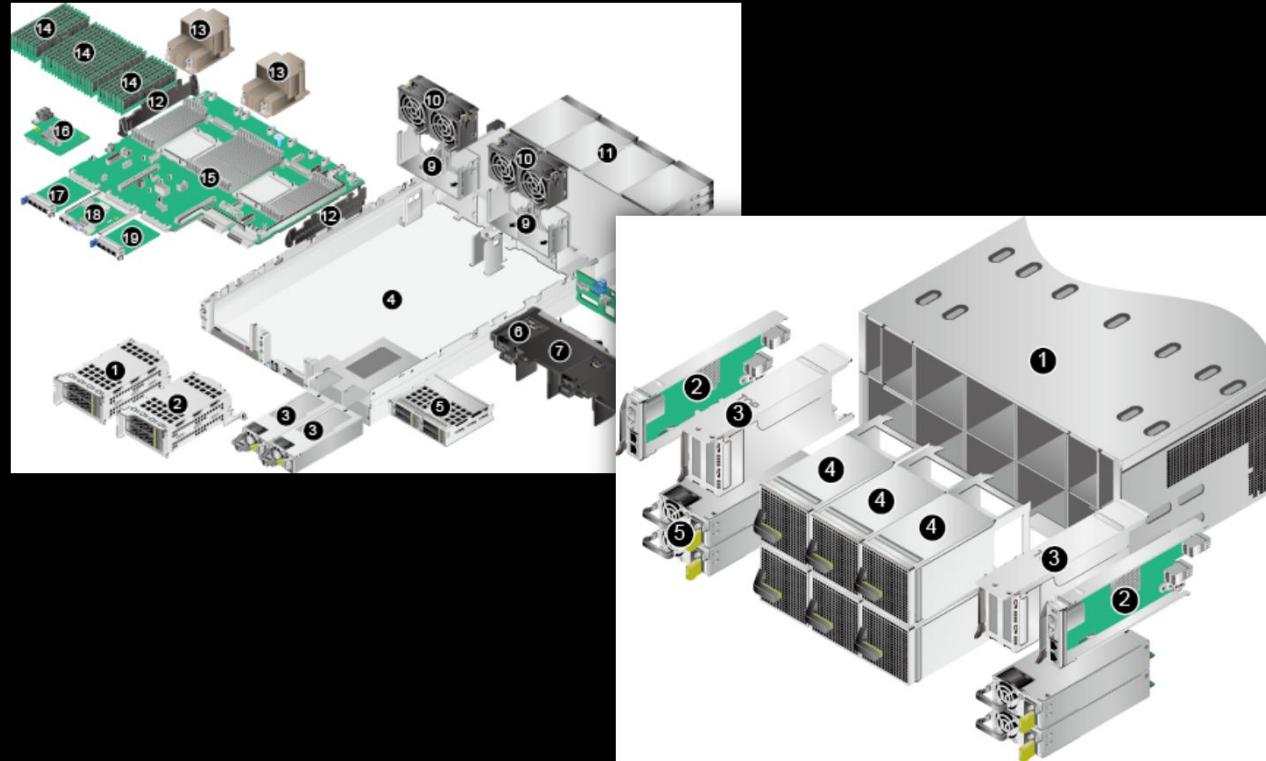


```
i2c@7000c000 {  
    compatible = "ascend, a910-i2c";  
    reg = ;  
    interrupts = ;  
    #address-cells = ;  
    #size-cells = ;  
    [...]  
};
```

构建时确定

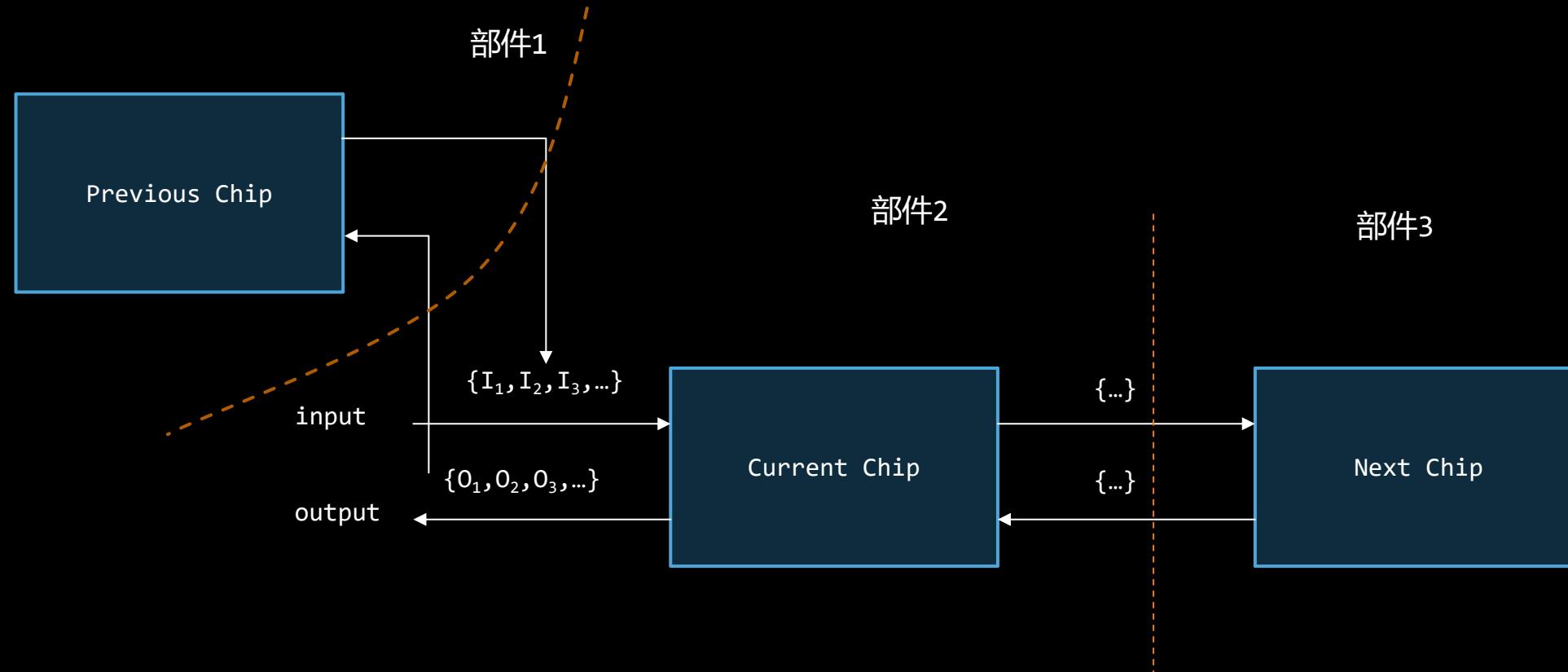
真实的系列化产品

真实的系列化产品 = 大量的公共部件 + 部件的组合



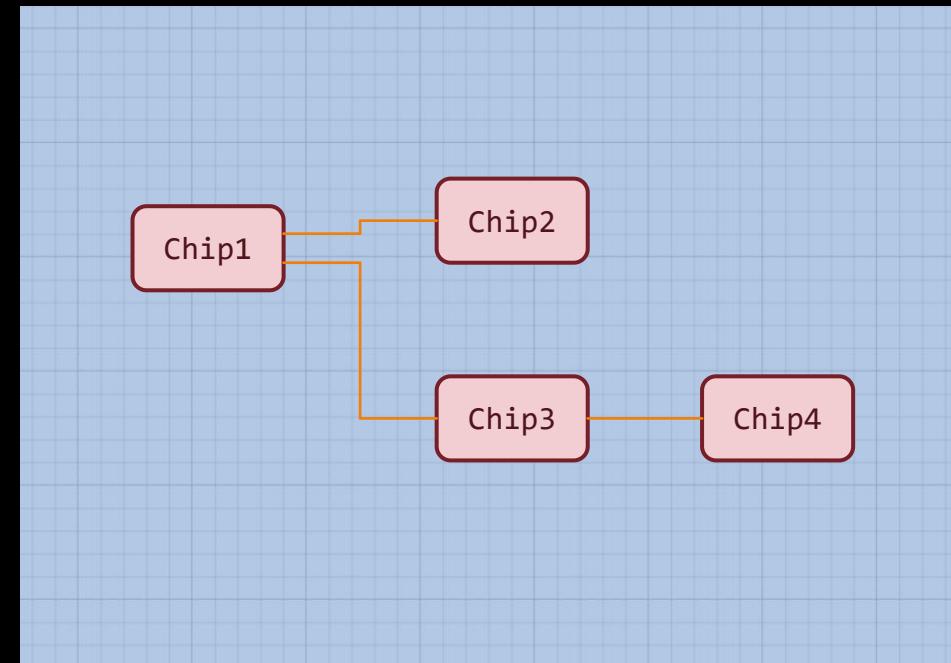
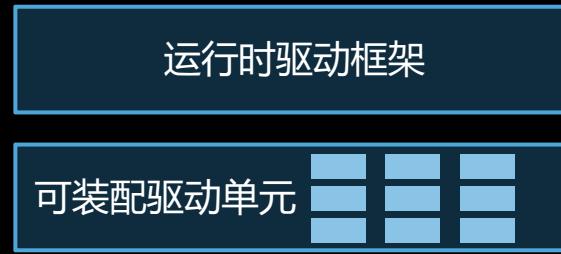
动态变化的电路拓扑

当器件的组合电路不确定时，驱动怎么写？



Runtime driver

驱动的行为在运行时而不是编译时确定

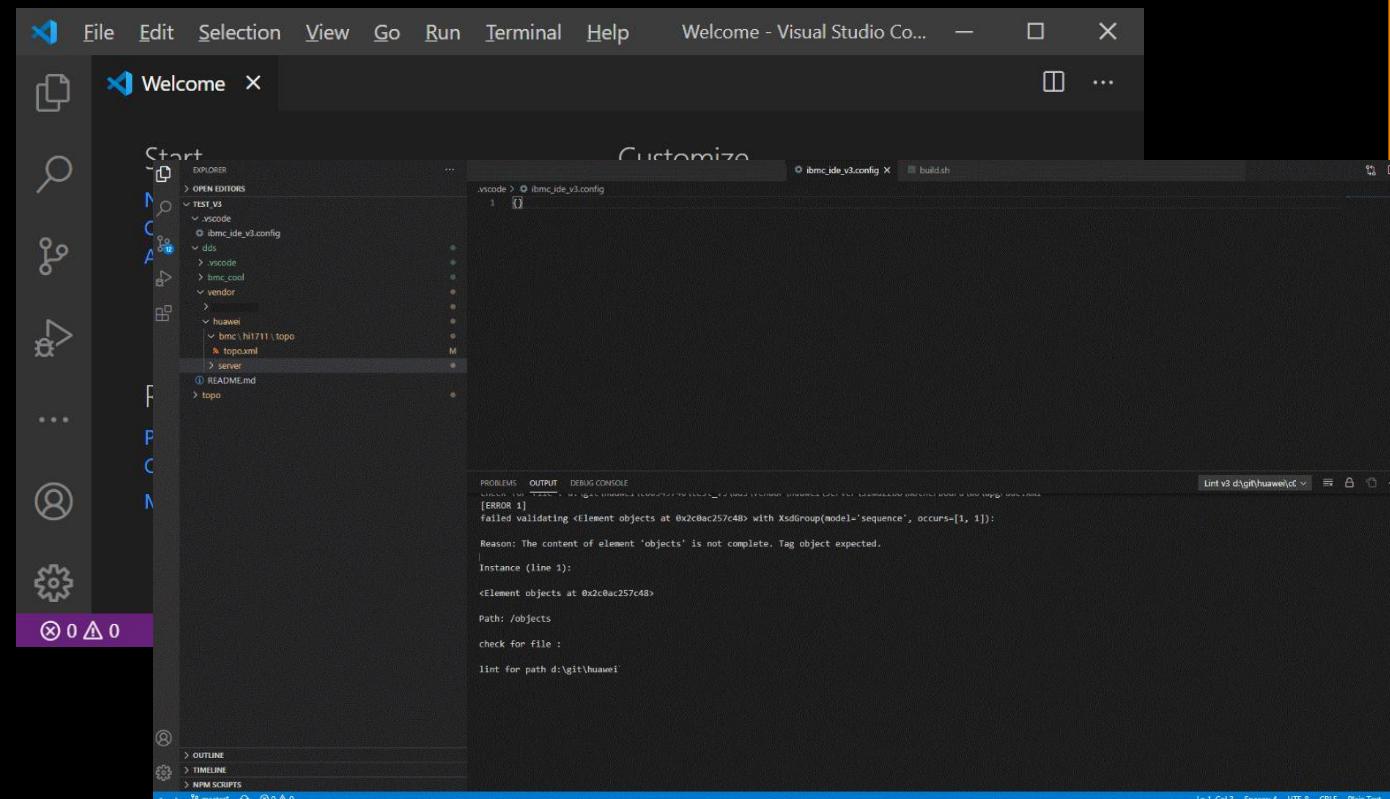


电路编辑工具

DSL IDE & Tool

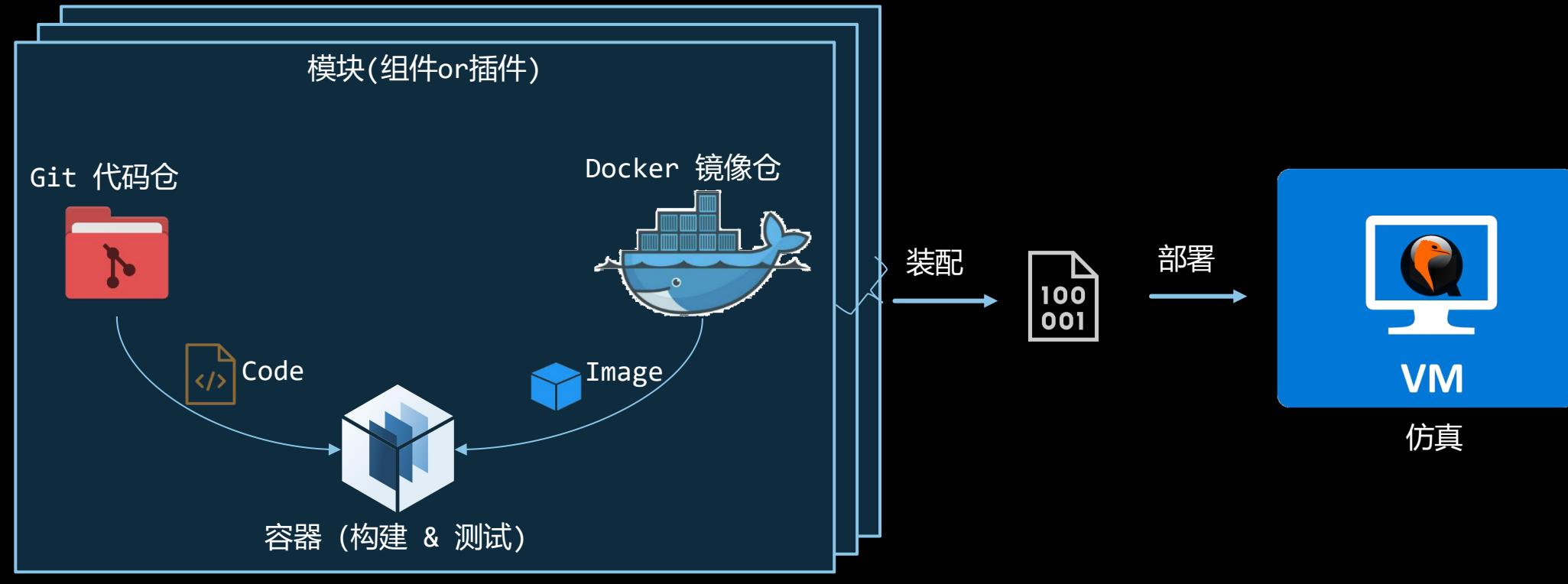
工具是效率的保障，好的工具让开发人员难以出错

- 语法检查
- 自动补齐
- 代码提示、跳转
- 依赖检查
- 代码自动生成



构建

- 并发构建，极速构建
- 二进制级别的装配



微组件架构总结

基于微组件架构可以获得以下收益：

- 可以以更细的粒度划分可独立开发的模块（组件及插件）
- 模块之间代码分仓隔离，以二进制文件形式进行组装
- 模块可以独立开展CI/CD，支持开发流水线
- 模块可以由不同的团队完成异步开发与测试
- 系统级的OOP，支持跨模块的多态性，支持接口的继承，支持重写与扩展
- 系统具备二进制级别的可裁剪性、可替换性
- 架构由DSL语言承载，业务代码支持混合语言，自由选择面向过程还是面向对象
- 接口的定义和业务逻辑代码分离
- 接口的看护可以工具化、可视化

Thank you.

华为鲲鹏®昇腾®计算出品

谈静国

华为ICT基础软件渗透测试技术专家



17年网络安全领域从业经验，擅长OS领域的漏洞挖掘和漏洞利用，擅长Linux内核提权、Android root等攻防技术。目前在华为从事自研OS、DB、编译器等基础软件的渗透测试工作。

主办方：

Boolan
高端IT咨询与教育平台

C++ Summit 2020

谈静国
华为ICT基础软件
渗透测试专家

渗透视角下的 C/C++安全编码 实践

目录

- 1.从一个漏洞说起 (CVE-2020-8597)
- 2.安全函数
- 3.静态扫描
- 4.fuzzing
- 5.安全编译选项
- 6.总结

CVE-2020-8597

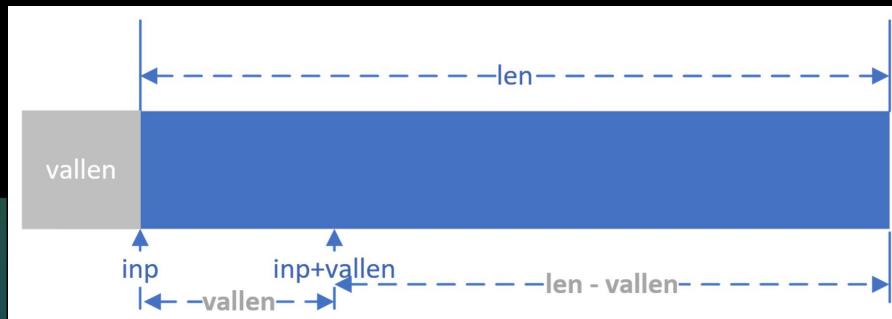
CVE-2020-5987 是pppd软件中存在17年之久的远程代码执行漏洞，CVSS评分为**9.8分(严重程度:Critical)**。

Ubuntu/Debian/Fedora等系统均受影响。

漏洞发生在处理Extensible Authentication Protocol (EAP)消息的eap.c:eap_request函数中。

如右图所示，inp中是网络传输的ppp数据，len和vallen均可控。

代码中存在逻辑错误，未判断**len - vallen**与**sizeof(rhostname)**大小，导致1429行发生缓冲区溢出。



```

1407:     case EAPT_MD5CHAP:
1408:     if (len < 1) {
1409:         error("EAP: received MD5-Challenge with no data");
1410:         /* Bogus request; wait for something real. */
1411:         return;
1412:     }
1413:     GETCHAR(vallen, inp);
1414:     len--;
1415:     if (vallen < 8 || vallen > len) {
1416:         error("EAP: MD5-Challenge with bad length %d (8..%d)",
1417:               vallen, len);
1418:         /* Try something better. */
1419:         eap_send_nak(esp, id, EAPT_SRP);
1420:         break;
1421:     }
1422:
1423:     /* Not so likely to happen. */
1424:     if (vallen >= len + sizeof (rhostname)) {
1425:         dbglog("EAP: trimming really long peer name down");
1426:         BCOPY(inp + vallen, rhostname, sizeof (rhostname) - 1);
1427:         rhostname[sizeof (rhostname) - 1] = '\0';
1428:     } else {
1429:         BCOPY(inp + vallen, rhostname, len - vallen);
1430:         rhostname[len - vallen] = '\0';
1431:     }

```

判断逻辑错误，永远为 false

rhostname为栈变量，栈溢出

CVE-2020-8597

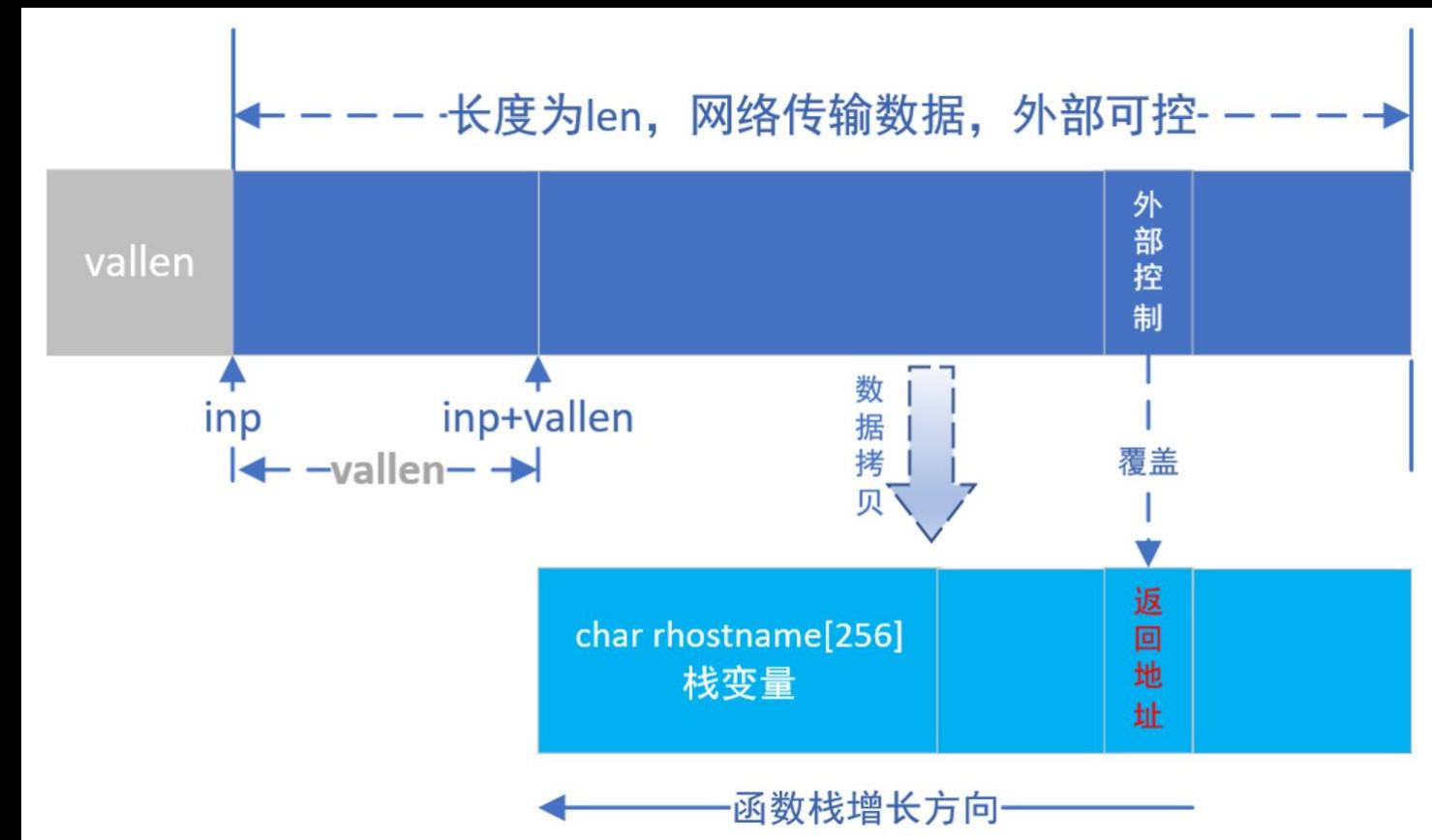
通过精心EAP数据包中的内容，在调用到BCOPY时覆盖函数**返回地址**：

`BCOPY(inp + vallen, rhostname, len - vallen);`

如右图示意图所示：

`char rhostname[256]; //栈上的变量`

函数返回地址覆盖之后，可以通过执行**ShellCode**或**ROP**等方式控制程序执行流，让pppd进程执行恶意代码。



CVE-2020-8597

漏洞修复patch：

```

✓ 4 pppd/eap.c □

↑ @@ -1420,7 +1420,7 @@ int len;
1420 } 1420 }
1421
1422     /* Not so likely to happen. */
1423 -     if (vallen >= len + sizeof (rhostname)) {
1424         dbglog("EAP: trimming really long peer name down");
1425         BCOPY(inp + vallen, rhostname, sizeof (rhostname) - 1);
1426         rhostname[sizeof (rhostname) - 1] = '\0';
1427     }
1428 @@ -1846,7 +1846,7 @@ int len;
1846 } 1846 }
1847
1848     /* Not so likely to happen. */
1849 -     if (vallen >= len + sizeof (rhostname)) {
1850         dbglog("EAP: trimming really long peer name down");
1851         BCOPY(inp + vallen, rhostname, sizeof (rhostname) - 1);
1852         rhostname[sizeof (rhostname) - 1] = '\0';
1853     }

```

修改错误的逻辑，将`if (vallen >= len + sizeof (rhostname))` 改为：

`if (len - vallen >= sizeof (rhostname))`

思考：这种方法是针对漏洞的专门修复，工程上有没有更普适的方法？

安全函数

漏洞关键代码：

```
BCOPY(inp + vallen, rhostname, len - vallen);
```

BCOPY是一个宏定义，没有长度判断，无保护机制：

```
#define BCOPY(s, d, l) memcpy(d, s, l)
```

使用安全函数memcpy_s：

```
memcpy_s(rhostname,sizeof(rhostname),  
         inp + vallen,len-vallen);  
//sizeof(rhostname) = 256
```

安全函数使用效果：

- 1) 无论len-vallen的值为多大，最多只往rhostname拷贝256字节，不会发生缓冲区溢出。
- 2) 逻辑漏洞未patch，也不会导致栈溢出。

安全函数

华为安全函数库（随openark等项目开源，mulan开源协议）：

https://gitee.com/openarkcompiler/OpenArkCompiler/tree/master/src/mapleall/huawei_secure_c/src

常见的安全函数包括：

- ✓ memcpy_s
- ✓ memmove_s
- ✓ memset_s
- ✓ scanf_s
- ✓ snprintf_s
- ✓ strcpy_s
- ✓ strncat_s
- ✓ strncpy_s
- ✓

跨平台支持：Windows/Linux

| | |
|----------------|-----------------------|
| ❑ fscanf_s.c | mv to mapleall folder |
| ❑ fwscanf_s.c | mv to mapleall folder |
| ❑ gets_s.c | mv to mapleall folder |
| ❑ input.inl | mv to mapleall folder |
| ❑ memcpy_s.c | mv to mapleall folder |
| ❑ memmove_s.c | mv to mapleall folder |
| ❑ memset_s.c | mv to mapleall folder |
| ❑ output.inl | mv to mapleall folder |
| ❑ scanf_s.c | mv to mapleall folder |
| ❑ secinput.h | mv to mapleall folder |
| ❑ secureutil.c | mv to mapleall folder |

安全函数

防止不正确的使用memcpy_s:

1) BCOPY(inp + vallen, rhostname, len - vallen); 

2) memcpy_s(rhostname,sizeof(rhostname),
inp + vallen,len-vallen); 

3) memcpy_s(rhostname, len-vallen,
inp + vallen, len- vallen); 

memcpy_s 2,4参数相同，未起到保护效果。

4) #define SAFE_COPY(d, s, l) memcpy_s(d, l, s, l)
SAFE_COPY(rhostname, inp + vallen, len-vallen) 

封装安全函数，导致memcpy_s 2,4参数相同，未起到保护效果。

可通过制定安全编程规范，禁止错误使用安全函数的行为。

整形溢出漏洞

Sample Code 1:

```
p = malloc(len + 1);
if(p != NULL) {
    memset(p,0,len);
}
```

漏洞在哪?

Sample Code 2:

```
p = malloc(nblocks * block_size);
if(p != NULL) {
    memset(p,0,blocks);
}
```

漏洞在哪?

整形溢出漏洞

Sample Code 1:

```
p = malloc(len + 1);      //len=0xffffffff len+1=0,len is unsigned int  
if(p != NULL) {  
    memset(p,0,len);    //heap overflow  
}  
溢出导致heap overflow.
```

Sample Code 2:

```
p = malloc(nblocks * block_size); //nblocks,block_size可控  
                                //nblocks = 0x10000 block_size = 0x10000  
                                //32位 nblocks*block_size = 0  
if(p != NULL) {  
    memset(p,0,blocks);  //heap overflow  
}  
溢出导致heap overflow.
```

OOB 漏洞

Sample Code 3:

```
int g_array[0x100];
```

```
int func(int idx) {  
    if(idx > 0x100) {  
        return -1;  
    } else {  
        return g_array[idx];  
    }  
}
```

漏洞在哪?

OOB 漏洞

Sample Code 3:

```
int g_array[0x100];

int func(int idx) {
    if(idx > 0x100) {
        return -1;
    } else {
        return g_array[idx];      //OOB if idx = -0x1000
    }
}
```

idx = -0x1000时，发生OOB(Out Of Bound).

使用安全函数无法避免整形溢出/OOB漏洞，工程上可采用什么方法防止？

静态代码扫描

常用静态代码扫描工具：

- ✓ Fortify
- ✓ CodeMars
- ✓ Coverity
- ✓ PinPoint
- ✓ CodeChecker
- ✓ LLVM Clang Static Analyzer (开源, 可定制checker插件)

通常使用静态代码扫描工具 + 定制分析规则通常能检测出漏洞：

Sample Code 1(整形溢出)

Sample Code 2(整形溢出)

Sample Code 3(OOB)

LLVM CSA

官方文档: <https://clang.llvm.org/docs/ClangStaticAnalyzer.html>

静态检测原理:

Call Graph + CFG + 符号执行约束求解

Sample Code 1:

```
int func(unsigned int len) {    //符号化:reg_$0<unsigned int len>
    p = malloc(len + 1);        //reg_$0<unsigned int len> + 1
    if(p != NULL) {
        memset(p, 0, len);
    }
}
```

对加法操作设置溢出检测条件 (在checker中实现) :

$\text{reg_\$0<unsigned int len>} > (\text{reg_\$0<unsigned int len>} + 1)$

进行约束求解, 若有解表示可以触发溢出; 若无解, 说明Call graph上有其他限制条件, 无法触发溢出。

LLVM CSA

LLVM CSA中已实现一些checker，如检查overflow的checker:

```
$ ./build/bin/clang -cc1 -analyzer-checker-help | grep overflow
alpha.security.ArrayBound      Warn about buffer overflows (older checker)
alpha.security.ArrayBoundV2    Warn about buffer overflows (newer checker)
alpha.security.MallocOverflow  Check for overflows in the arguments to malloc()
alpha.unix.OverFlow            Check for overflow.
```

检查越界读、越界写的checker:

```
$ ./build/bin/clang -cc1 -analyzer-checker-help | grep Bound
alpha.security.ArrayBound      Warn about buffer overflows (older checker)
alpha.security.ArrayBoundV2    Warn about buffer overflows (newer checker)
alpha.unixcstring.OutOfBounds  Check for out-of-bounds access in string functions
osx.coreFoundation.containers.OutOfBounds
```

问题：成因比较复杂的漏洞无法检测，如race condition，需要自定义实现checker插件。

Race Condition-> Double Free

Sample Code 4: (Race condition 导致 Double Free)

```
int delete_data(int index) {
    DATA_INFO* pinfo;
    pthread_mutex_lock(&g_mutex);
    pinfo = search_for_data(index);      //A,B两个线程均通过相同index找到相同pinfo
    pthread_mutex_unlock(&g_mutex);

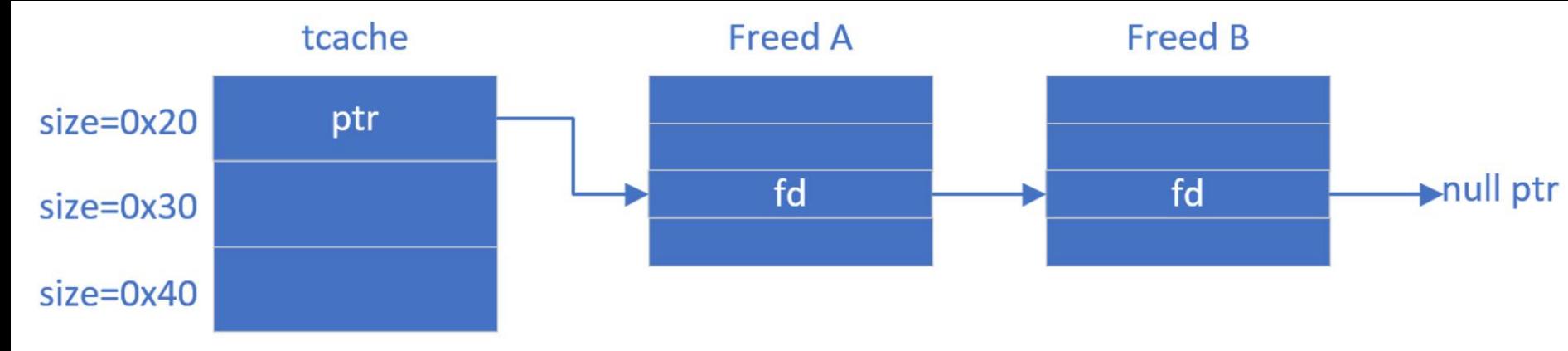
    pthread_mutex_lock(&g_mutex);
    if(pinfo!= NULL) {
        free_data(pinfo);            //A,B两个线程先后调用free_data(pinfo),Double Free
        pinfo = NULL;
    }
    pthread_mutex_unlock(&g_mutex);

}
```

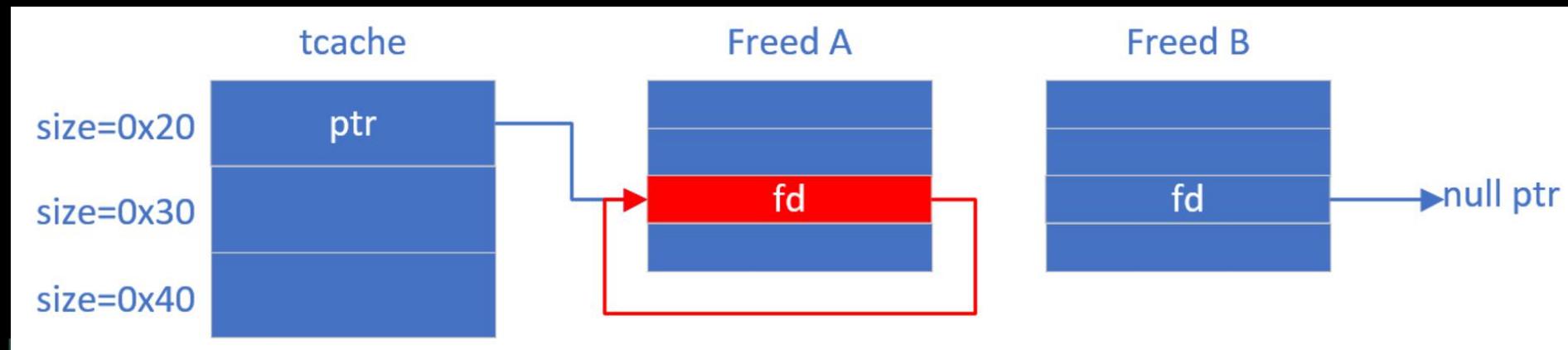
Race Condition --> Double Free造成的危害是怎样?

Double Free的危害

Double Free可以转换为任意地址写任意值，进而可执行恶意代码，获取Shell。
glibc较新版本中引入tcache，free释放的A B两块内存挂在tcache链表上：

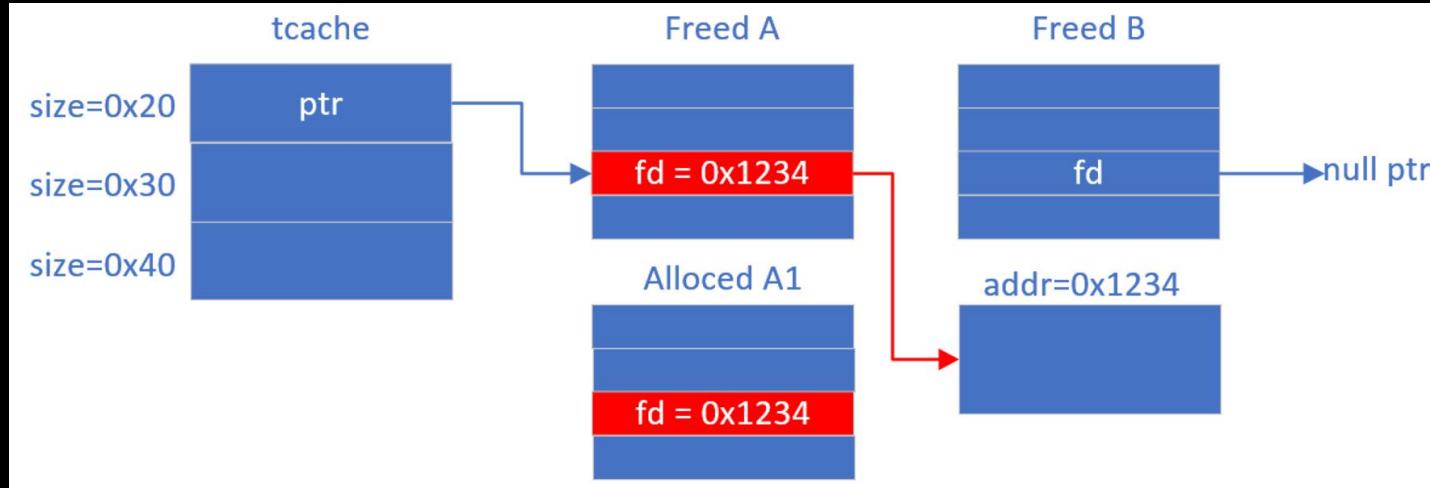


Double Free发生时，再次free A，链表结构被破坏，成为自循环：

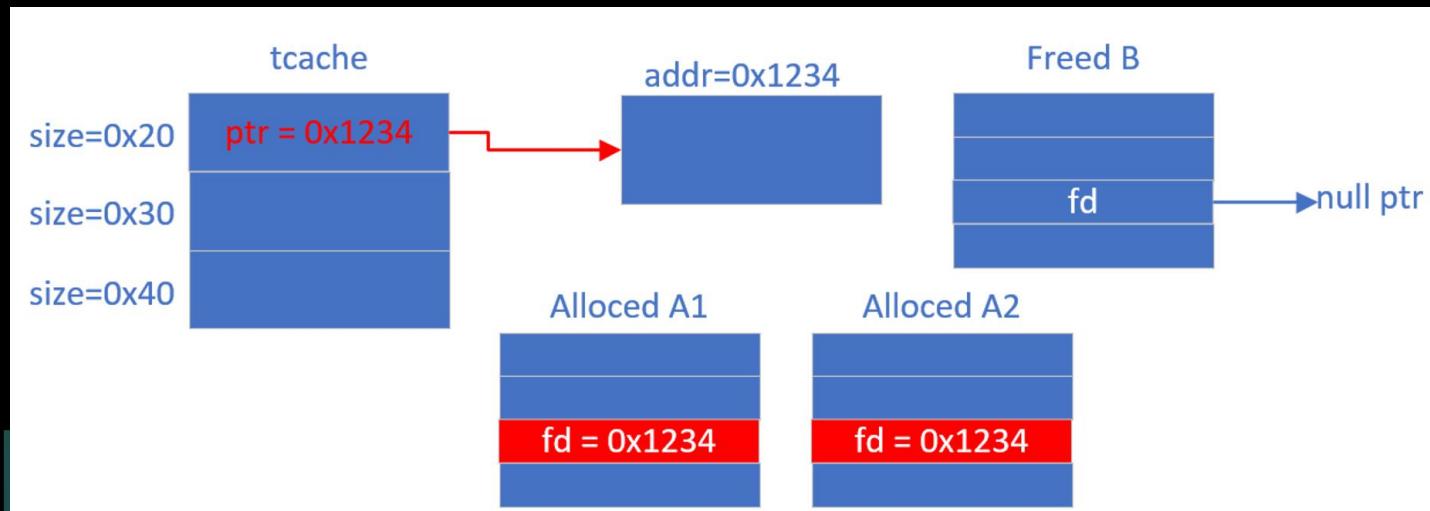


Double Free的危害

申请一块内存Alloced A1后，tcache的ptr仍然指向Freed A，Alloced A1 = Freed A
写A1内存fd = 0x1234(或其他任意地址)

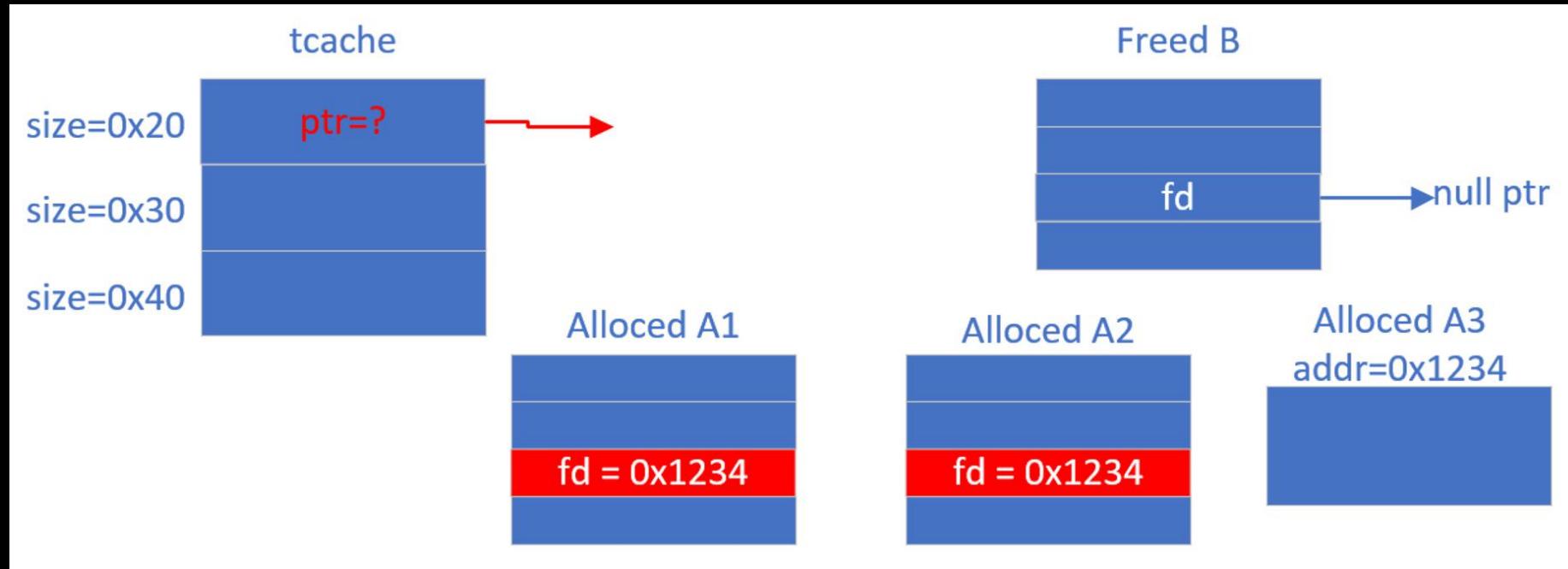


再次申请一块内存A2之后，tcache ptr指向0x1234 (Allocated A1 = Allocated A2)



Double Free的危害

再申请一块内存，申请到的Alloced A3地址为0x1234（或其他任意地址）。



此时往Alloced A3中写入数据，形成任意地址写任意值。

Double Free的危害

具备任意地址写任意值能力之后，向全局变量`__malloc_hook`地址写入OneGadget地址，再次调用`malloc`时，glibc中`__libc_malloc`将执行OneGadget地址的指令，如下所示：

```

3034: void *
3035: __libc_malloc (size_t bytes)
3036: {
3037:     mstate ar_ptr;
3038:     void *victim;
3039: |
3040:     void *(*hook) (size_t, const void *)
3041:     = atomic_forced_read (__malloc_hook);
3042:     if (__builtin_expect (hook != NULL, 0))
3043:         return (*hook)(bytes, RETURN_ADDRESS (0));
3044: #if USE_TCACHE

```

OneGadget地址执行`execve("/bin/bash")`，如下所示：

```

→ test one_gadget /lib/x86_64-linux-gnu/libc.so.6
0x4f3d5 execve("/bin/sh", rsp+0x40, environ)
constraints:
    rsp & 0xf == 0
    rcx == NULL

```

Race Condition检测

Race Condition LLVM CSA漏洞检测建模：

- 1) 识别访问共享资源。定义访问共享资源操作，如访问链表 list_head等可认为是访问共享资源。
- 2) 识别使用锁。定义锁操作，如pthread_mutex_lock等函数。
- 3) 识别存在Race Condition的行为。

规则：Call graph中一条路径执行过程中，多组lock/unlock访问共享资源，且多组访问之间无额外锁保护，则认为存在Race Condition问题。

```
int delete_data(int index) {  
    DATA_INFO* pinfo;  
    pthread_mutex_lock(&g_mutex);    //1.1 .识别到第一组lock/unlock  
    pinfo = search_for_data(index); //2 search_for_data函数中链表操作，识别到访问共享资源  
    pthread_mutex_unlock(&g_mutex); //1.2 识别到第一组lock/unlock  
  
    pthread_mutex_lock(&g_mutex);    //3.1 识别到第二组lock/unlock  
    if(pinfo!= NULL) {  
        free_data(pinfo);           //4 free_data函数中链表操作，识别到访问共享资源  
        pinfo = NULL;  
    }  
    pthread_mutex_unlock(&g_mutex); //3.2 识别到第三组lock/unlock  
}
```

LLVM CSA

测试CSA checker:

存在问题的函数`delete_data`, 在62行查找`pinfo`, 在67行释放。

两个线程并发时, 62行均找到`pinfo`, 两次调用67行释放`pinfo`, Double Free。

运行结果提示`delete_data -> free_data`这条调用路径存在Double Free, 如下所示:

```
#0 Calling free_data at line 67
#1 Calling delete_data
Double Free Detected...
./sample/race_double_free.c:53:5: warning: Call Stack:#0 Calling free_data at line 67#1 Calling delete_data
    list_del(&(pinfo->list));
^~~~~~
2 warnings generated.
```

其他类型的漏洞如UAF等也可以通过建模方式检测。

```
32 DATA_INFO* search_for_data(int index)
33 {
34     DATA_INFO* pinfo;
35     list_for_each_entry(pinfo,&g_data_head,list)
36     {
37         if(index == pinfo->index)
38         {
39             return pinfo;
40         }
41     }
42     return NULL;
43 }

51 int free_data(DATA_INFO* pinfo)
52 {
53     list_del(&(pinfo->list));
54     free(pinfo);
55     return 0;
56 }

58 int delete_data(int index)
59 {
60     DATA_INFO* pinfo;
61     pthread_mutex_lock(&g_mutex);
62     pinfo = search_for_data(index);
63     pthread_mutex_unlock(&g_mutex);
64
65     pthread_mutex_lock(&g_mutex);
66     if(pinfo != NULL) {
67         free_data(pinfo);
68         pinfo = NULL;
69     }
70     pthread_mutex_unlock(&g_mutex);
71     return 0;
72 }
```

Fuzzing

Fuzzing能够发现更多的安全编码漏洞。

Fuzzing也是一门博大精深的学问。

常用的fuzzer有：

- ✓ libfuzzer (基于代码覆盖率变异，开发阶段使用)
- ✓ hongfuzz
- ✓ SecCodeFuzz
- ✓ Trinity
- ✓ AFL
- ✓ Peach
- ✓ SecDive
- ✓ syzkaller
- ✓ ...

Fuzzing时加上 ASAN，更好的探测到内存错误。

```
int stack_overflow(const uint8_t *Data, size_t Size) {  
    char buf[32];  
    memcpy(buf, Data, Size);  
    return 0;  
}  
  
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {  
    stack_overflow(Data, Size);  
    return 0;  
}
```

libfuzzer函数级fuzz用例

安全编译

通过安全编译，可使漏洞无法利用，或大幅提升漏洞利用的难度。

几个常用的安全编译选项：

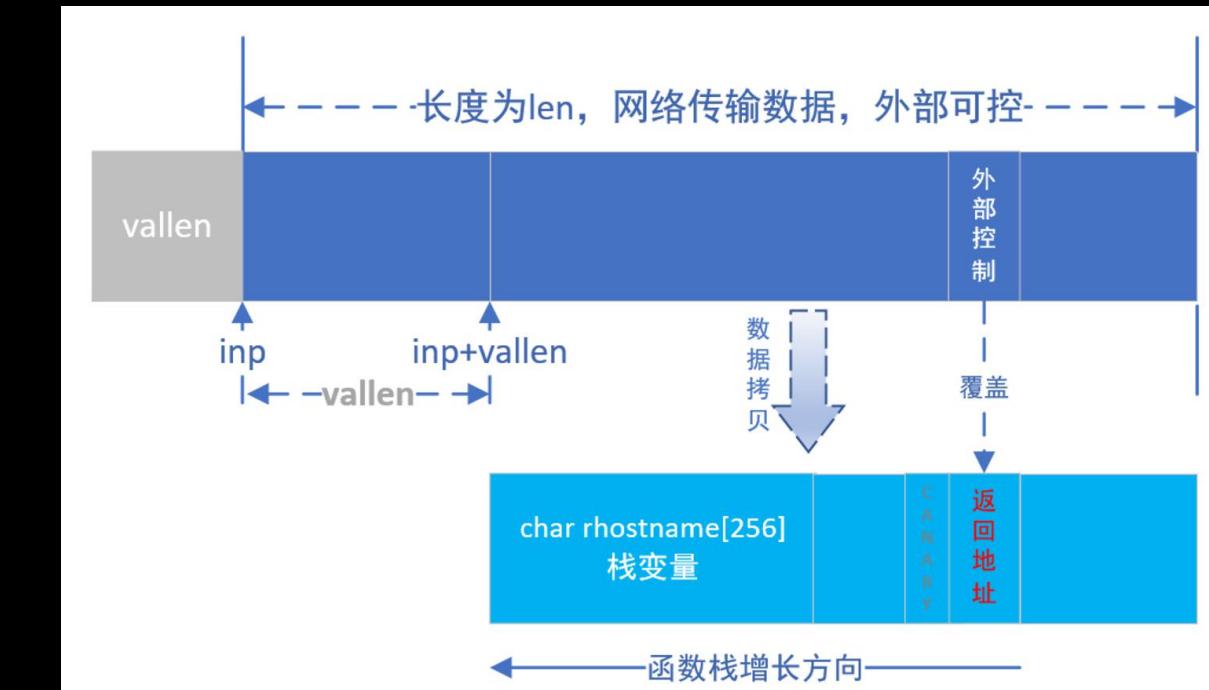
- NX 堆栈不可执行
防止注入shellcode到堆栈并执行

- SP 栈保护
生成stack canary，探测到栈溢出时crash

- PIE 地址无关
系统支持ASLR时程序各segment地址随机化

地址随机化之后，攻击者无法获取程序代码段地址，无法获取libc加载地址增加攻击难度。
例如：Double Free利用成功必须要先获取libc的加载地址。

- Strip
删除符号表，增大程序被逆向分析的难度。（对开源软件无影响）
- RELRO
GOT表保护，Full RELRO时无法修改GOT表。



总结

开发安全可信的代码，工程上可采用的方法：

✓ 安全函数库

推荐！在不改变程序逻辑的情况下避免漏洞，但要防止使用方法错误和错误封装

✓ 安全编程规范

约束员工编码行为，编写可信的代码

✓ 商用静态代码扫描工具

商用工具 + 定制规则 解决一部分问题

✓ 定制静态代码扫描工具

基于LLVM + CSA，通过漏洞建模，编写自定义扫描工具增强发现能力

✓ Fuzzing + ASAN

通过模糊测试发现漏洞

✓ 安全编译选项

在漏洞尚未修复的情况下可使漏洞无法利用或增大漏洞利用的难度

Q&A

王博

资深技术咨询师



资深技术咨询师。专注于大型系统软件的领域建模设计、重构、持续交付以及服务化、智能化架构演进等领域，在技术咨询合作过程中同时帮助企业进行软件人员技能和开发效率提升工作。曾深度参与大规模嵌入式软件平台、无线通讯4G、5G核心系统、自动驾驶、云存储、人工智能对话平台等技术咨询项目。

主办方：

Boolan
高端IT咨询与教育平台

CPP-Summit 2020

王博
软件技术咨询师

多范式融合的 Modern C++ 软件设计

议程

1

C++编程范式演进概览

2

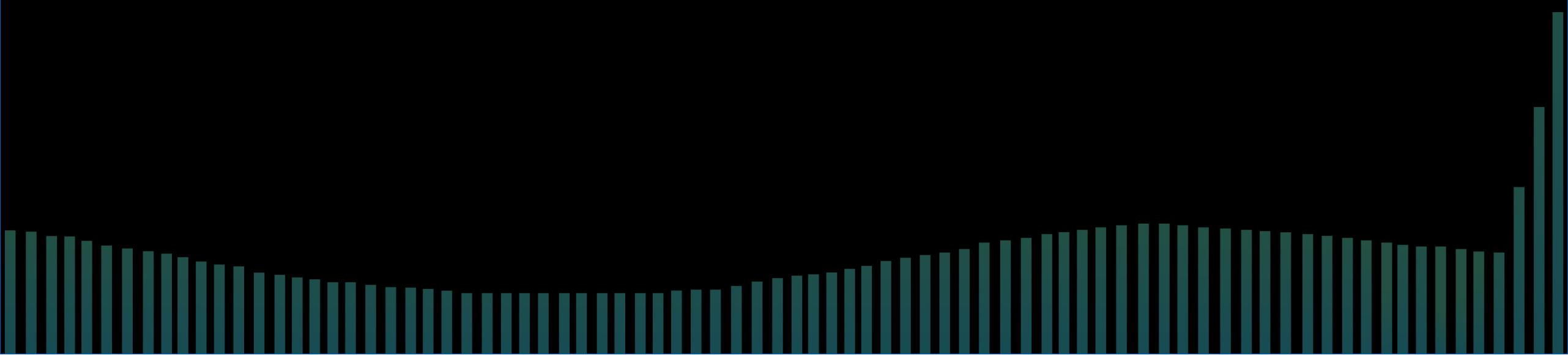
C++编程范式剖析

3

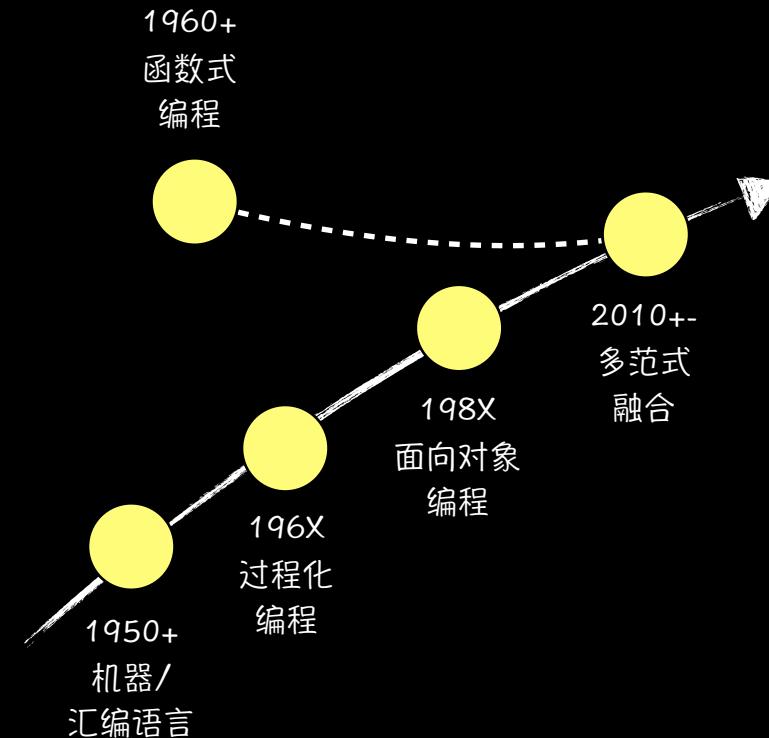
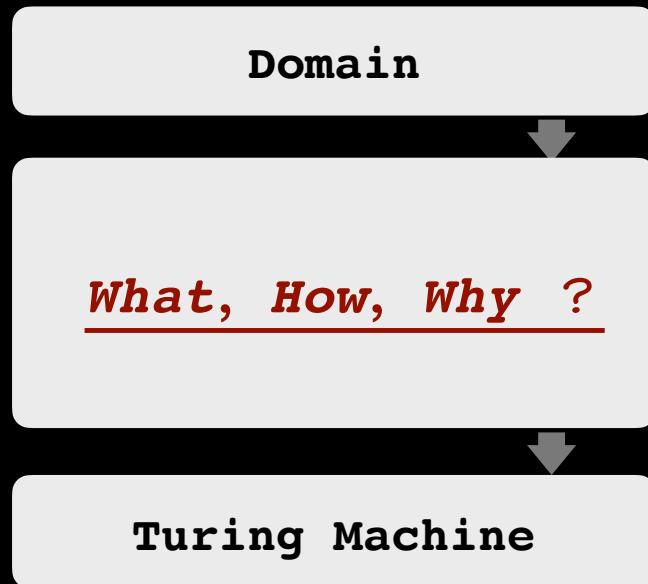
多范式融合的C++程序设计策略

01

C++编程范式演进概览



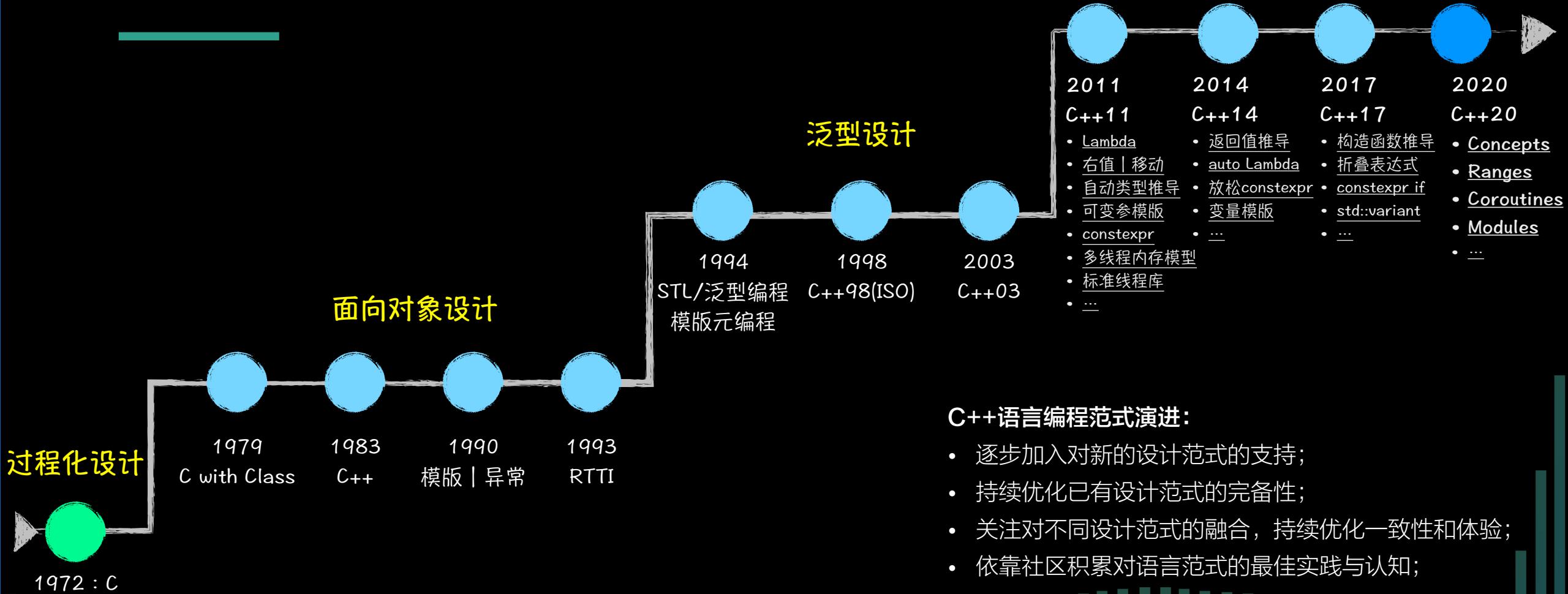
编程范式



编程范式 (Programming Paradigm) :

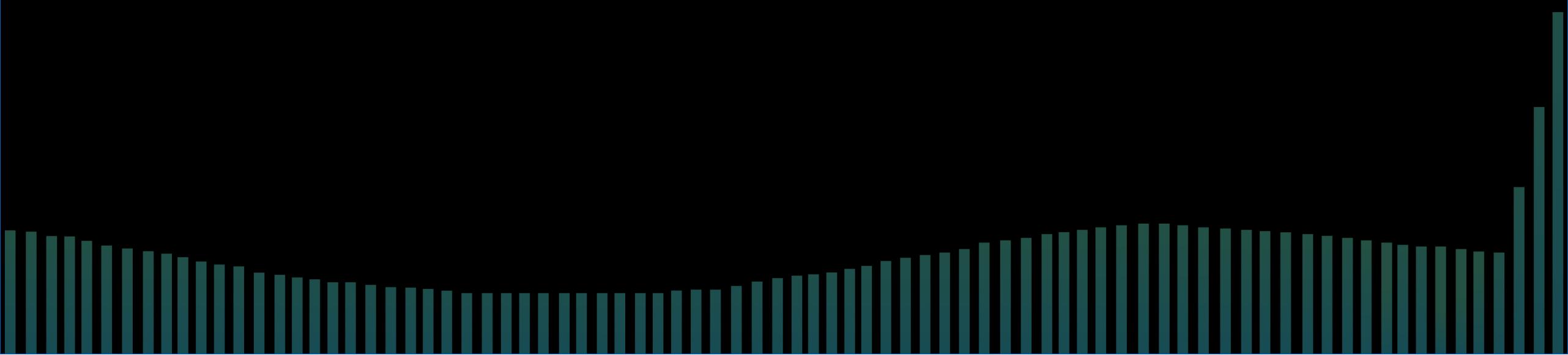
- 最早出自Robert Floyd的1979图灵奖颁奖演说；
- 是程序员看待程序的观点，认为程序如何被设计的基本思维；
- 编程范式和软件建模方式及架构风格有紧密关系；

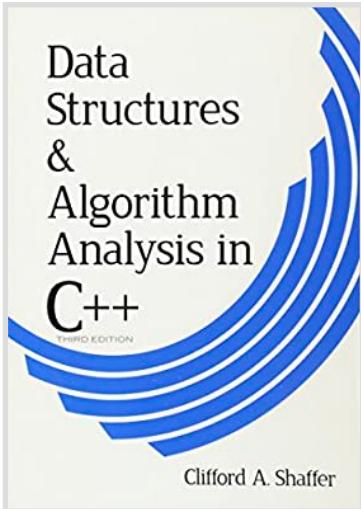
C++语言编程范式演进



02

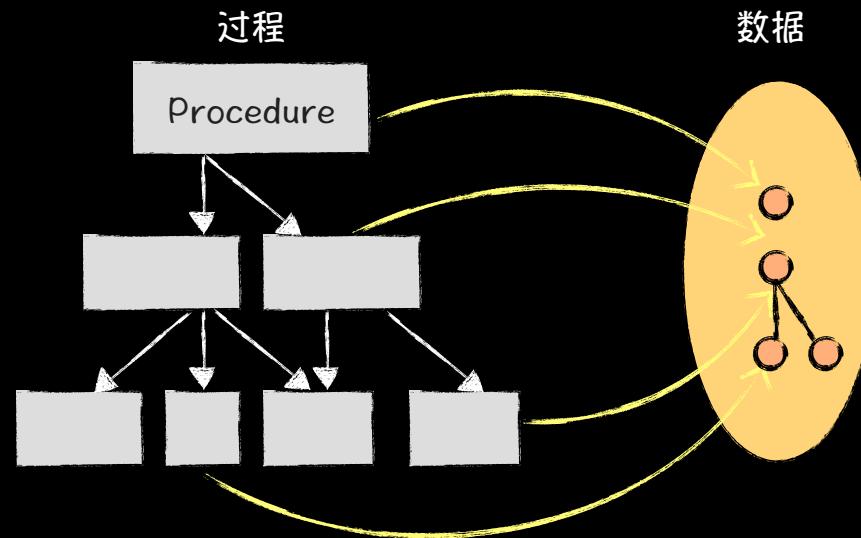
C++编程范式剖析





过程式设计

过程式设计



- 过程式编程
 - “程序 = 数据结构 + 算法”

Nicklaus Wirth

优点：贴近图灵机模型，充分调动硬件，可控性强；

缺点：数据的全局可访问性带来较高耦合复杂度，局部可复用性及响应变化能力较差；

软件设计对确定性的追求



静态确定性：

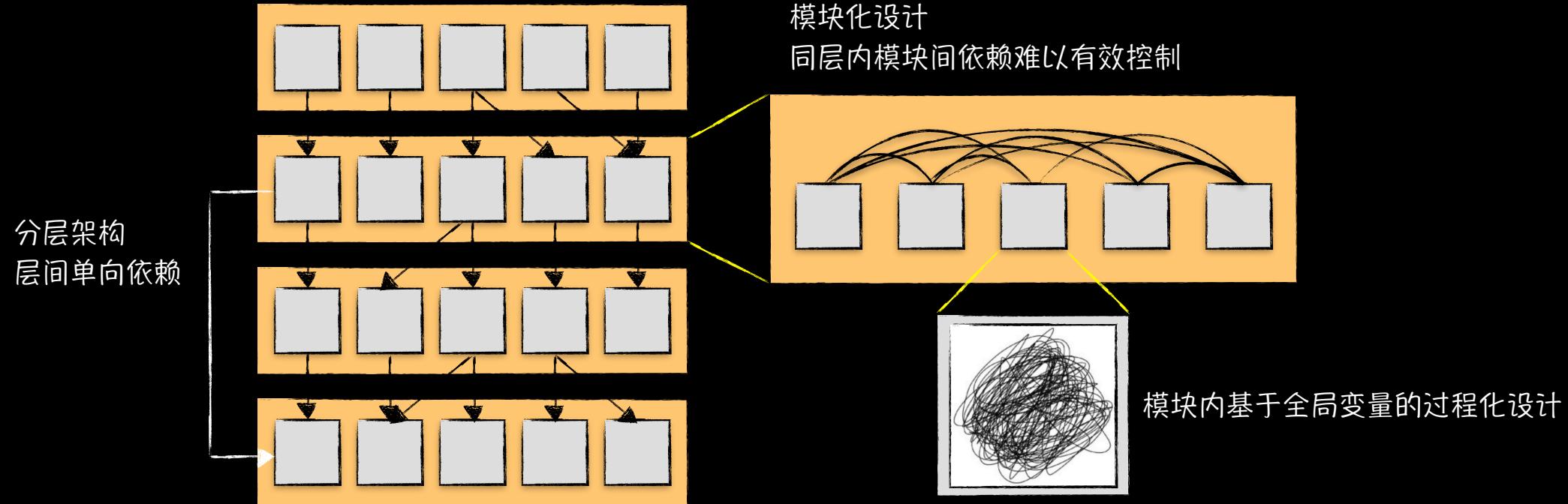
- 代码符号与ABI的确定性控制：name mangling追溯，可控inline，库的接口兼容性控制…
- 二进制大小的确定性控制：template显示具现、disable RTTI、disable exception…



动态确定性：

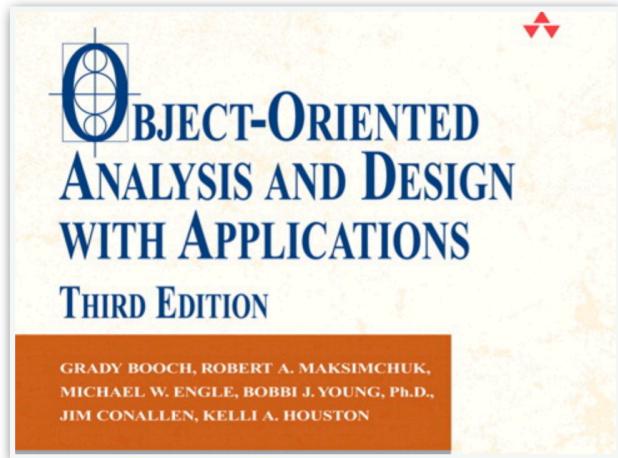
- 内存使用的确定性控制：placement_new, alignof, std::align, std::aligned_storage, std::pmr::memory_resource …
- 运行时指令执行的确定性控制：disable RTTI、disable exception、多线程内存模型…

过程式设计与分层模块化架构风格



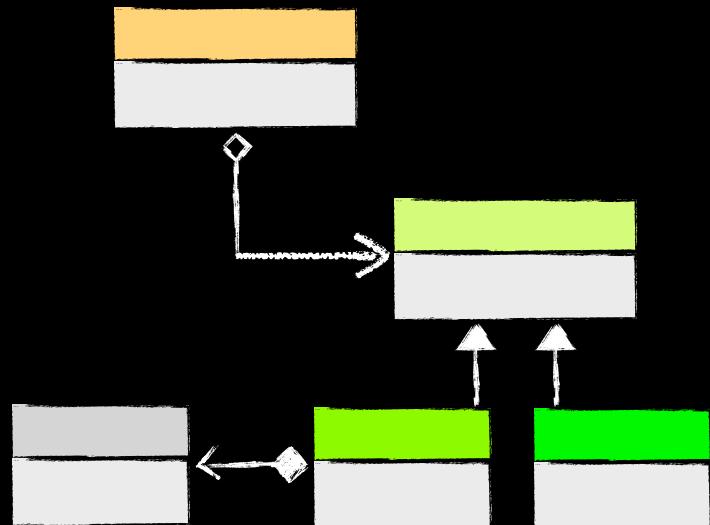
架构风格：分层次、划模块、定接口，设计数据结构（数据建模）、设计算法流程；

特点：支撑了大规模并行开发，偏静态规划式开发交付；可裁剪性和可复用性过粗，响应变化能力相对较弱；



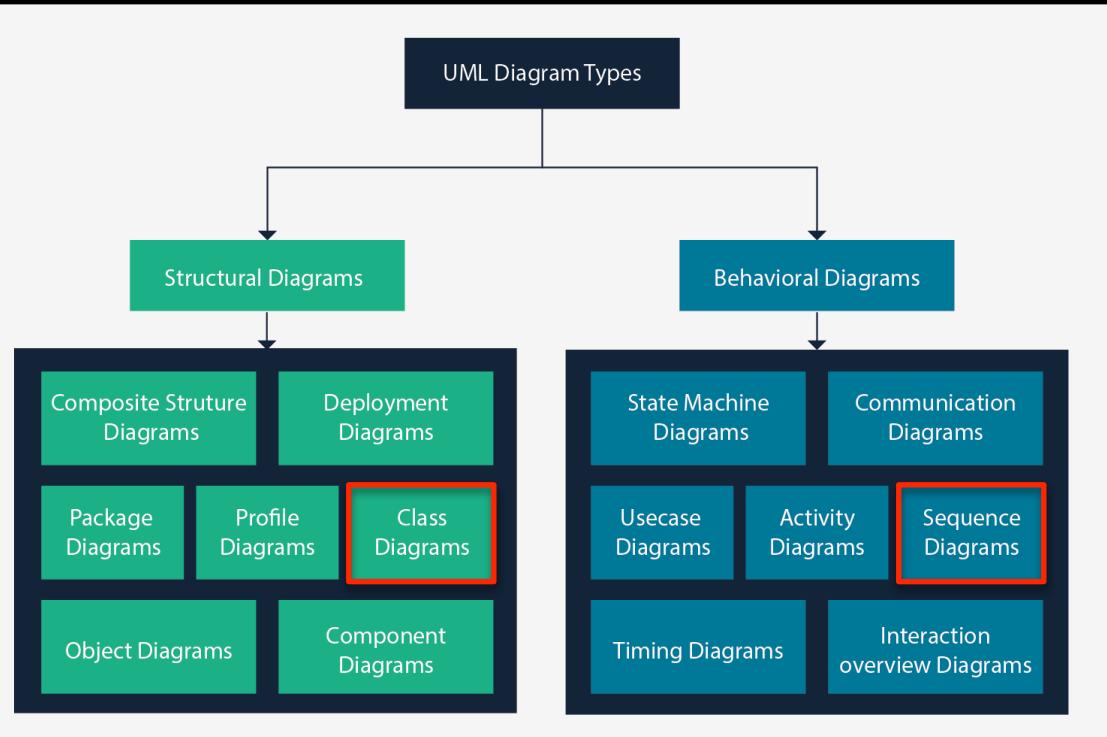
面向对象设计

面向对象设计

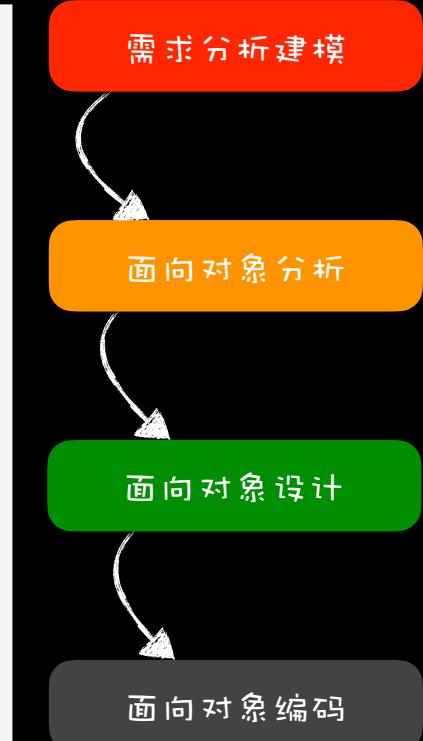


- 面向对象编程
 - 程序 = 实体 + 关系
 - 核心特点：封装，继承与多态

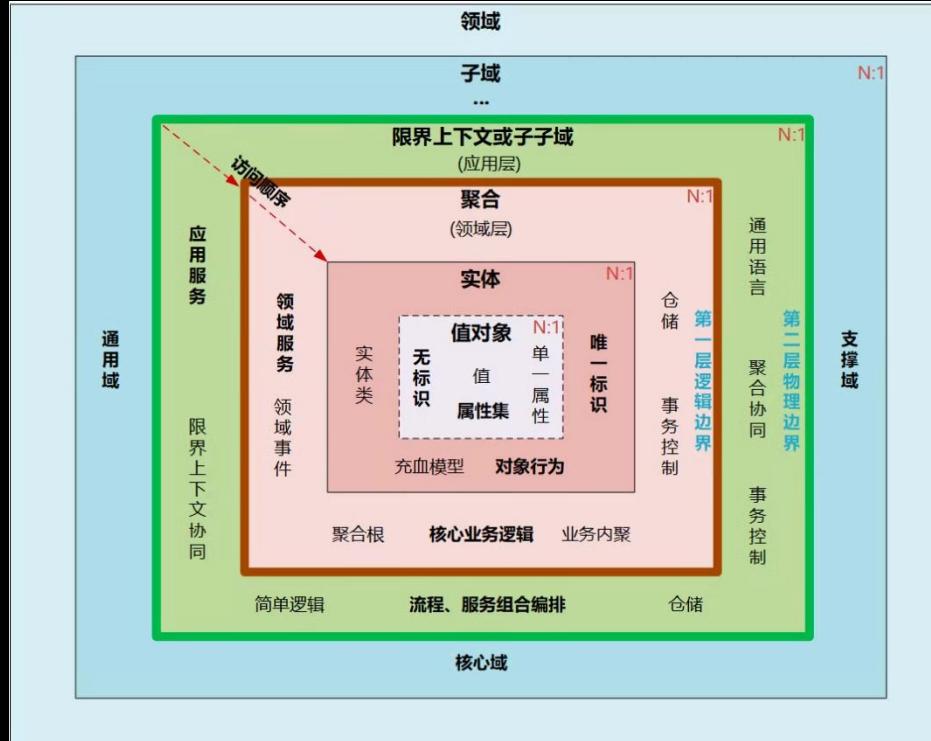
面向对象建模



静态视图：类 + 结构



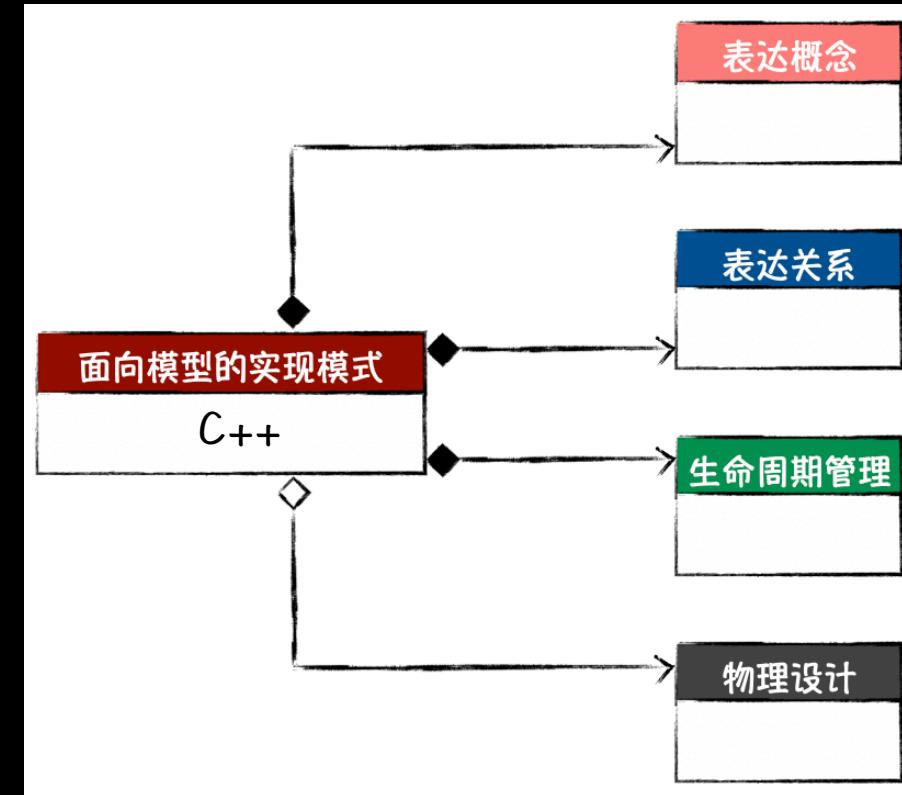
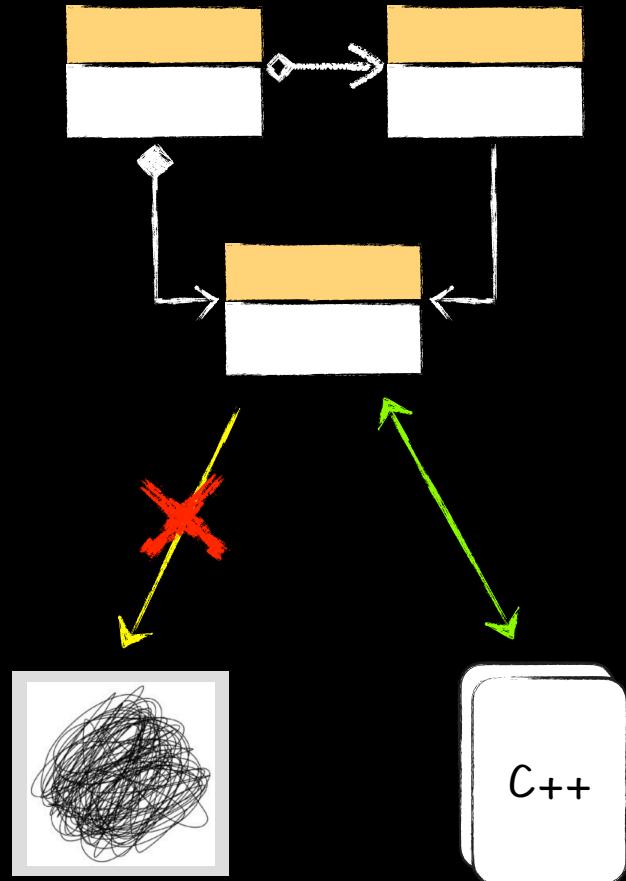
动态视图：对象 + 交互



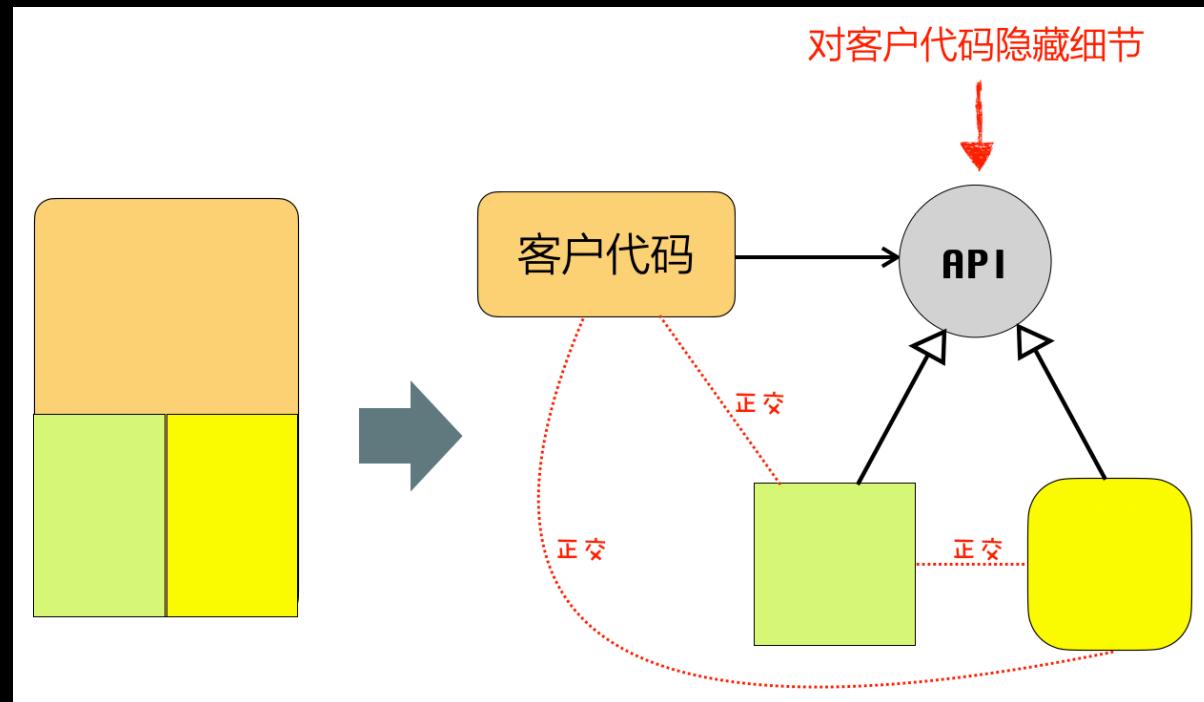
面向对象建模

领域驱动设计

面向模型C++实现模式



从设计原则到设计模式



SOLID原则（核心是对多态的设计原则）

- **S: 单一职责原则：**（职责为变化的原因）
通过接口分离变和不变，隔离变化；
- **O: 开放封闭原则；**
多态的目标在于系统对于变化的扩展而非修改；
- **L: 里氏替换原则；**
接口设计要达到细节隐藏的圆满效果；
- **I: 接口隔离原则；**
面向不同客户的接口要分离开；
- **D: 依赖倒置原则；**
接口的设计和规定者应该是接口的使用方；

掌握C++的各种实现模式

C++多态:

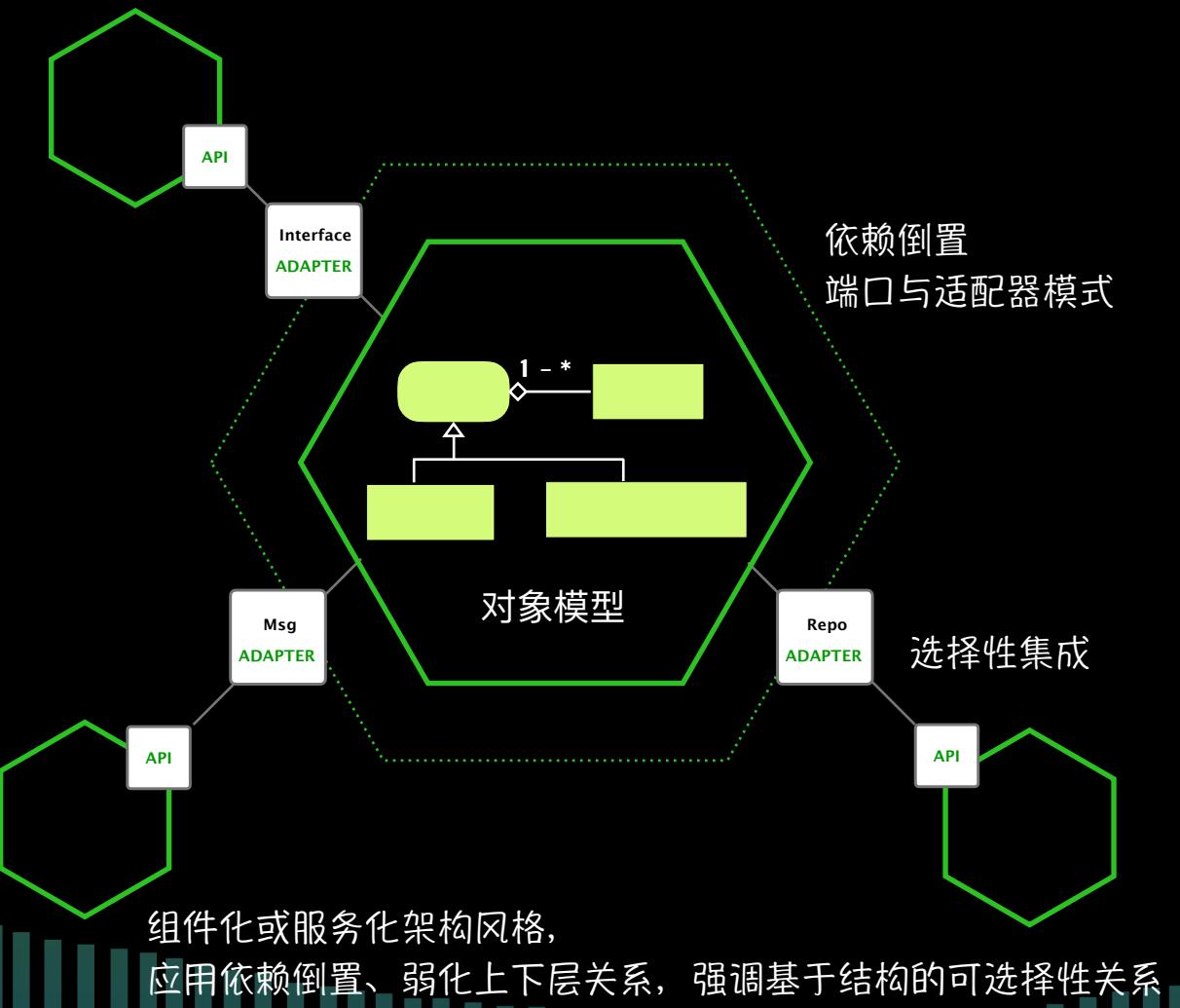
- 静态多态: 函数重载、泛型编程
- 动态多态: 函数指针、虚函数、类型擦除

```
struct Triangle {
    void draw() const {
        std::cout << "△" << std::endl;
    }
};

struct Circle {
    void draw() const {
        std::cout << "○" << std::endl;
    }
};

TEST_F(draw_shape) {
    using Shape = std::variant<Triangle, Circle>;
    std::vector<Shape> shapes {Circle{}, Triangle{}};
    for (const auto &shape : shapes) {
        std::visit([](const auto &s) { s.draw(); }, shape);
    }
}
```

面向对象设计的架构风格

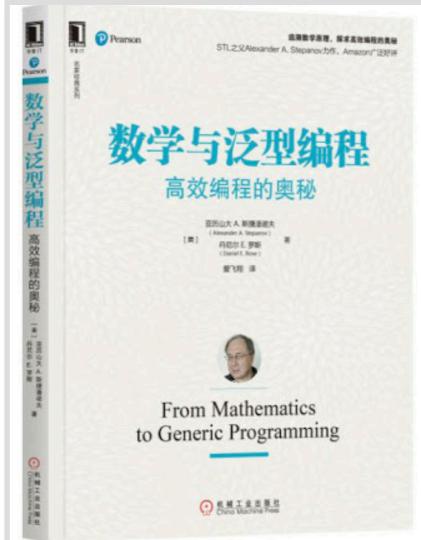


优点:

- 对象自封装数据和行为，利于理解和复用；
- 对象作为“稳定的设计质料”，适合泛领域使用；
- 多态提高了响应变化的能力，进一步提升了软件规模；
- 对设计的理解和演进优先是对模型和结构的理解和调整；

不足:

- 业务逻辑碎片化，散落在离散的对象内；
- 行为和数据的失匹配协调（贫血与充血） \rightarrow DCI架构；
- 面向对象建模依赖工程经验，缺乏严格的理论支撑；



泛型编程

泛型编程是解决类型变化的手段

代码中的三种变化：

- 数据变化：代码中变化的是行为所操作属性和数据不同
- 行为变化：代码中变化的是局部行为操作的不同
- 类型变化：代码中变化的是操作所针对的类型的不同

换种视角：三种类型变化都可以看作是数据变化

接口是对数据类型的可操作性契约的显式化表达；

- 1) 将变化参数化；
- 2) 将参数契约化；
(客户可以执行哪些合法操作)
- 3) 将契约接口化；

稳定部分面向接口契约编程；

变化部分面向接口契约扩展；

上下文根据场景选择满足接口契约的具体实现进行组合；

```
template<EqualityComparable Equal>
void process(Object* obj[], size_t size, Handler&, Equal&&) {
}

// contract of Object instances
struct Object {
    void action();
    int a,b;
};

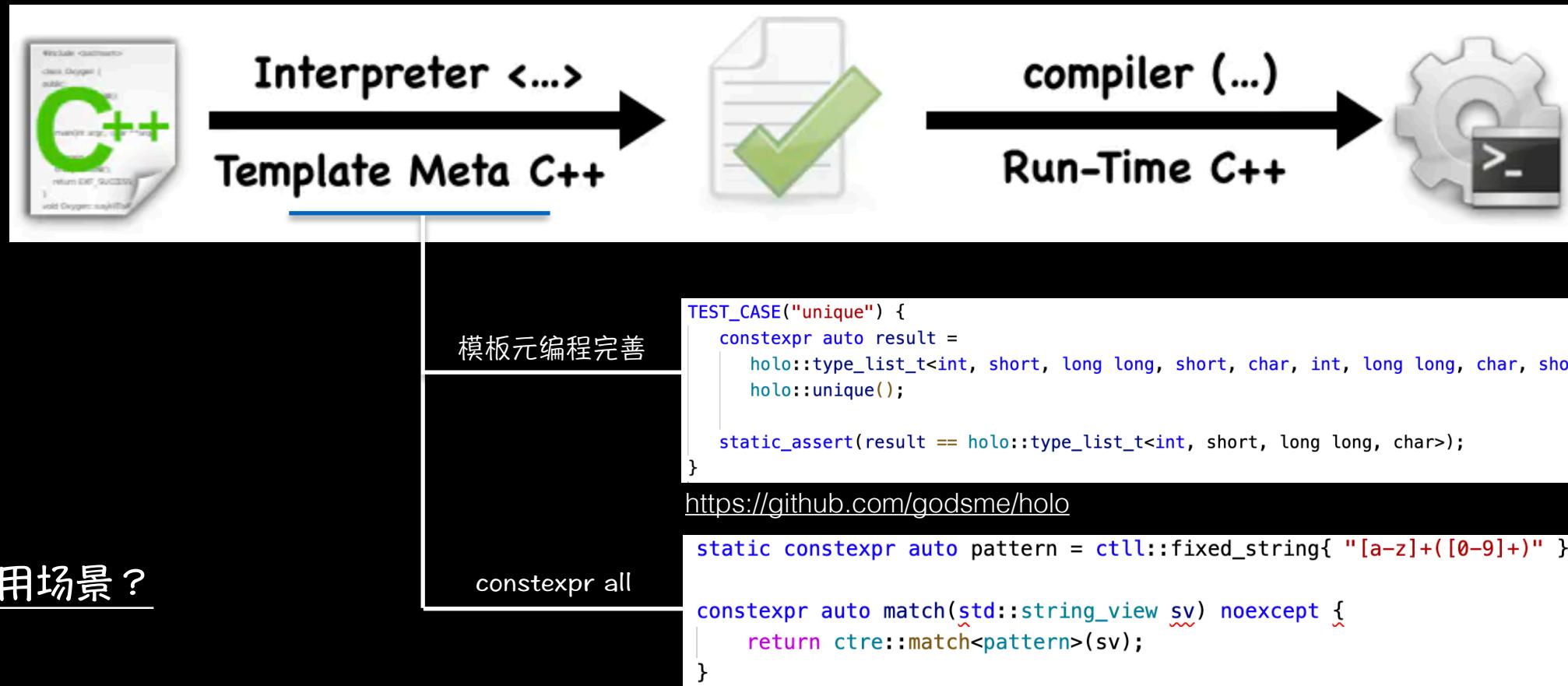
// contract of Handler implements
class Handler {
    virtual void handle(const Object& obj) = 0;
    virtual ~Handler(){}
};

// contract of EqualityComparable types
template<typename T>
concept bool EqualityComparable = requires(T a, T b) {
    {a==b}=>bool; {a!=b}=>bool; {b==a}=>bool; {b!=a}=>bool;
};
```

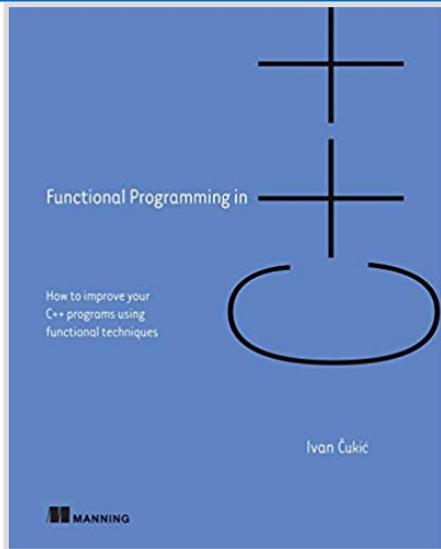
将变化数据化，给了对变化进行封装、持有、转移、延迟调用的能力（表现在间接性带来的抽象能力增强）

从泛型编程到编译期计算

- 两阶段的C++语言：编译期计算，运行期计算； (<https://www.jianshu.com/p/b56d59f77d53>)

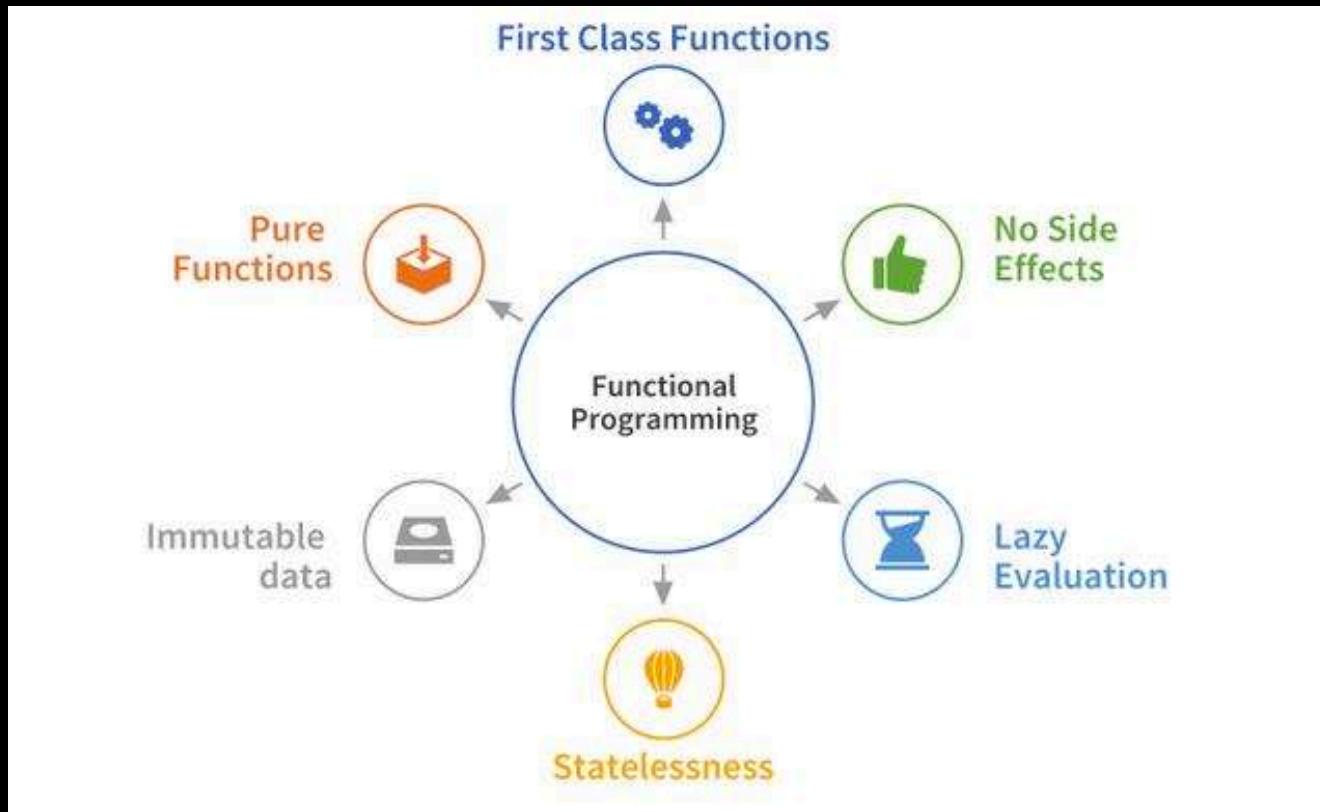


最佳使用场景？



函数式编程

函数式设计



- 函数式设计
 - “程序 = 数据 + 映射（函数）”

Lambda in C++

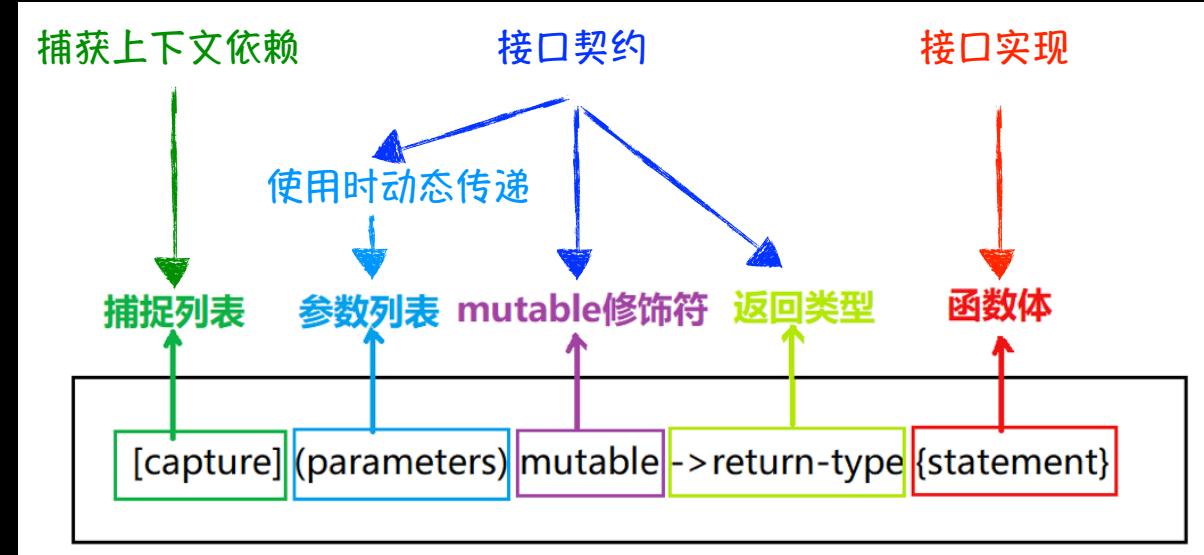
```

    客户期望的调用接口原型
    //////////////////////////////////////////////////////////////////
void process(std::function<int(int)> accumulator) {
    int ret = accumulator(5);
}

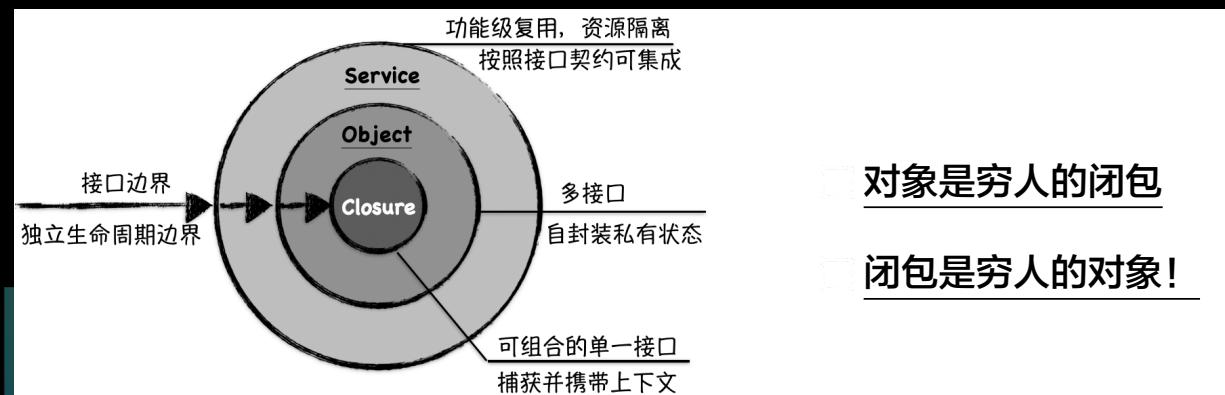
    声明需要捕获的上下文
class Functor {
public:
    Functor(int context) : member(context) {
    }
    int operator()(int parameter) {
        // implementation
        return 0;
    }
private:
    int member;
};

int context = 0;
process(Functor(context));
    根据环境捕获上下文，创建实例

    声明并根据环境捕获上下文，创建实例
process([context](int factor)->int {
    // implementation
    return 0;
});
    接口定义，内部具体实现
  
```



- 闭包: 独立生命周期、捕获上下文、单一接口、可组合
- Lambda让闭包更低成本的被使用，请不要因此滥用lambda！！！



函数式设计：高阶函数与组合（抽象代数）

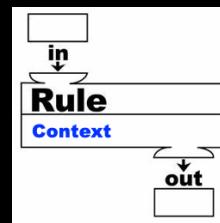
OO pattern/principle

- Single Responsibility Principle
- Open/Closed principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Factory pattern
- Strategy pattern
- Decorator pattern
- Visitor pattern

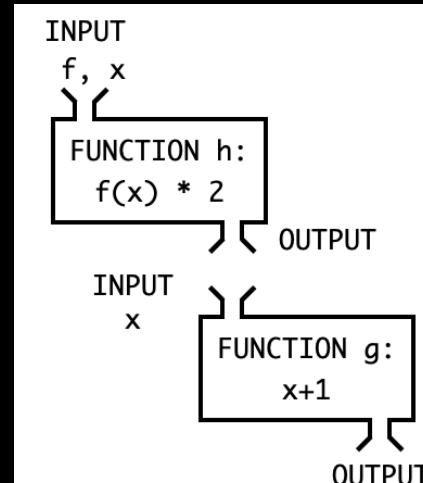
FP pattern/principle

- Functions
- Functions
- Functions, also
- Functions
- Yes, functions
- Oh my, functions again!
- Functions
- Functions ☺

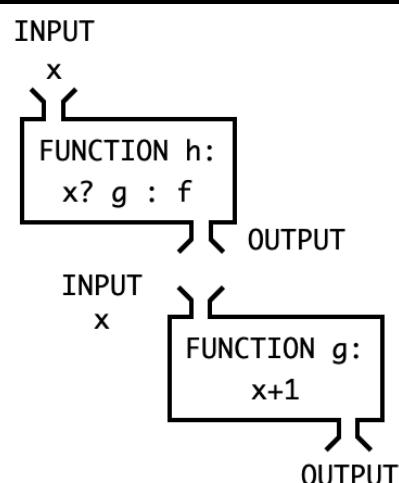
Seriously, FP patterns are different



借助闭包的单一接口的标准化，
以及高阶函数的可组合性，
通过规则串联设计，
完成数据从源到结果的映射描述！

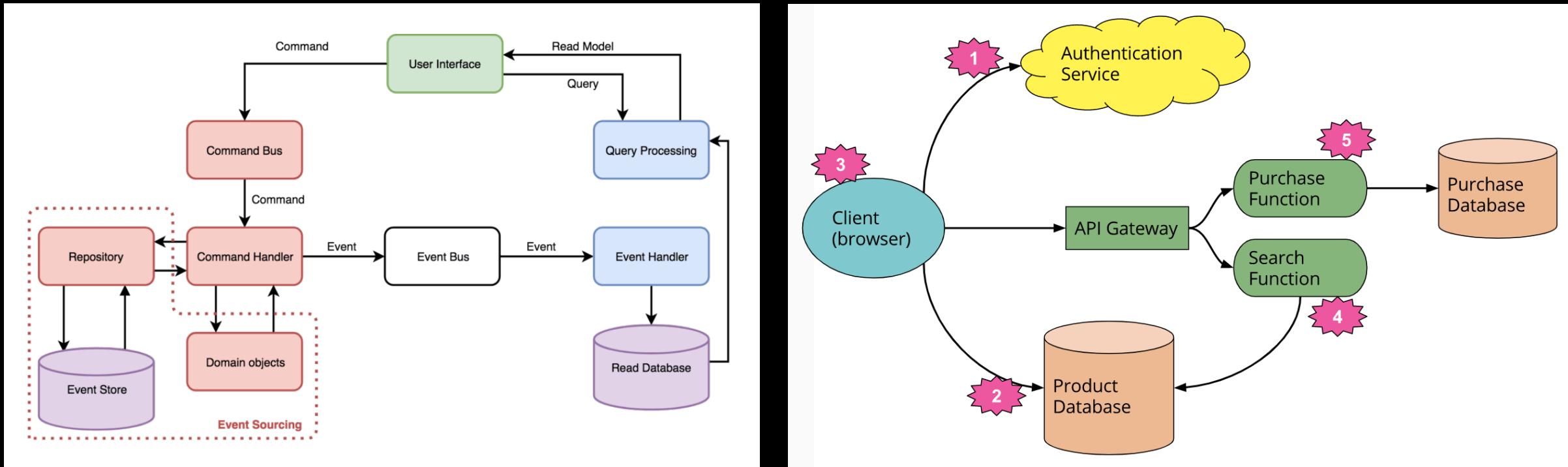


$$h(g, 2)=6$$



$$h(1)(3)=4$$

借鉴了函数式设计的架构风格



借鉴了函数式范式的架构风格:

- Event Sourcing、Reactive Architecture
- Lambda Architecture、FaaS、Serverless;

C++中的函数式设计

优点:

- 高度的抽象，易于扩展；
- 声明式表达，易于理解；
- 形式化验证，易于自证；
- 不可变状态，易于并发；

不足:

- 对问题域的代数化建模门槛高，适用域受限；
- 在图灵机上性能较差；
- 不可变的约束造成了数据泥团耦合；
- 闭包往往接口粒度过细；需要再组合才能构成业务概念

■ <https://github.com/ReactiveX/RxCpp>

```
auto threads = rxcpp::observe_on_event_loop();

// infinite (until overflow) stream of integers
auto values = rxcpp::observable<>::range(1);

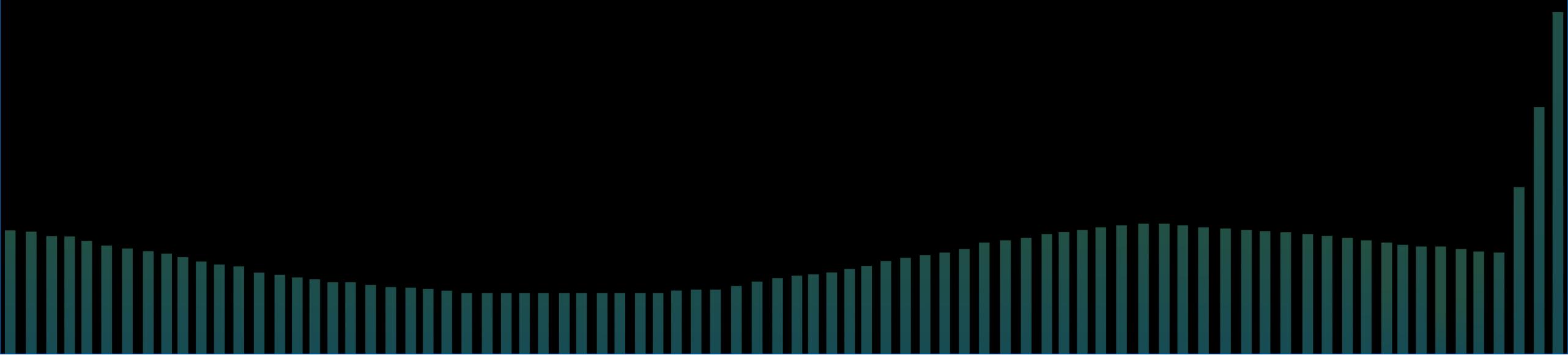
auto s1 = values.
    subscribe_on(threads).
    map([](int prime) {
        std::this_thread::yield();
        return std::make_tuple("1:", prime);});

auto s2 = values.
    subscribe_on(threads).
    map([](int prime) {
        std::this_thread::yield();
        return std::make_tuple("2:", prime);});

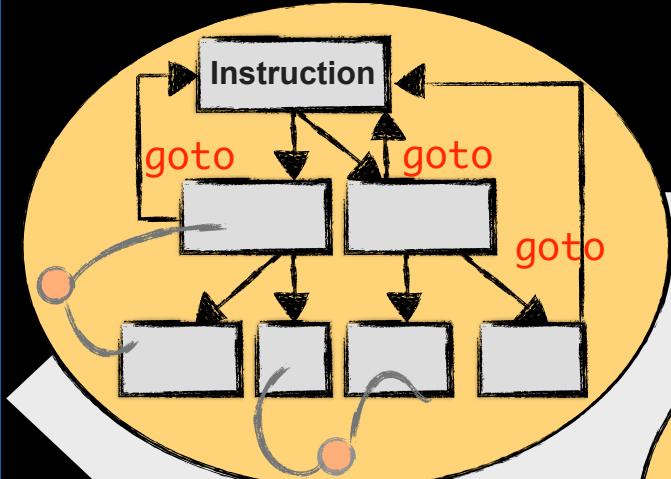
s1.
    merge(s2).
    take(6).
    observe_on(threads).
    as_blocking().
    subscribe(rxcpp::util::apply_to(
        [](<const char* s, int p) {
            printf("%s %d\n", s, p);
        }));
```

03

多范式融合的C++程序设计策略



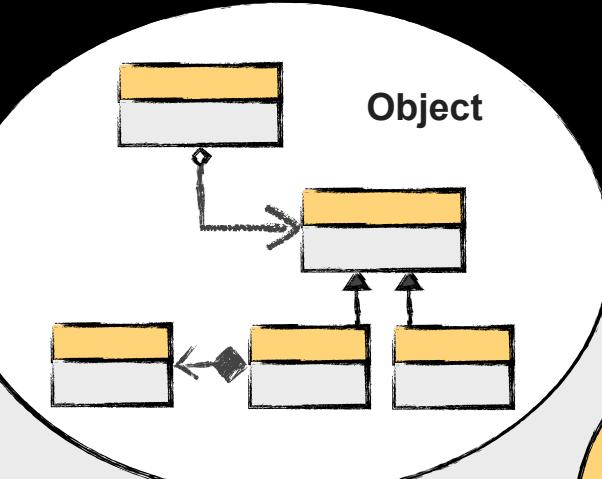
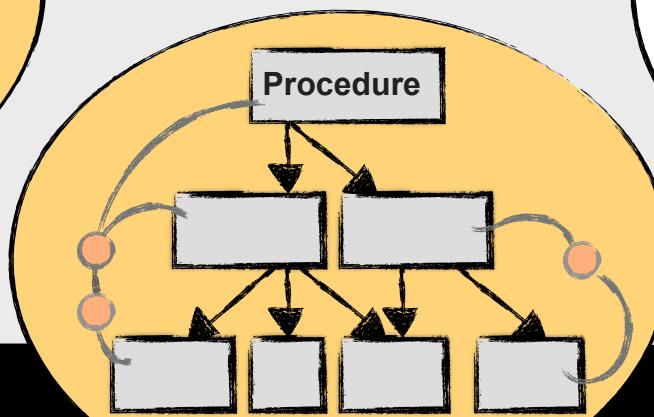
编程范式关系



约束了指令的方向性

数据与流程建模

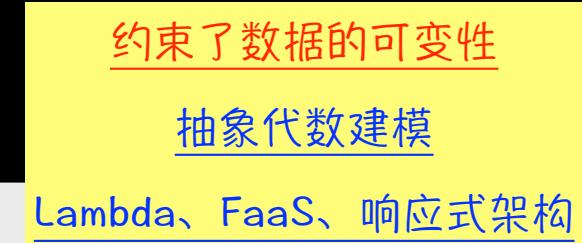
分层的模块化架构



约束了数据的作用域

面向对象分析与设计建模

组件化、服务化架构



约束了数据的可变性

抽象代数建模

Lambda、FaaS、响应式架构

越往左边限制越少，越贴近图灵机模型，可以充分调动硬件；“直接”带来的可控性及广域适用性；

越往右边限制越多，通过约束建立规则，通过规则描述系统；“抽象”带来的定域扩展性；

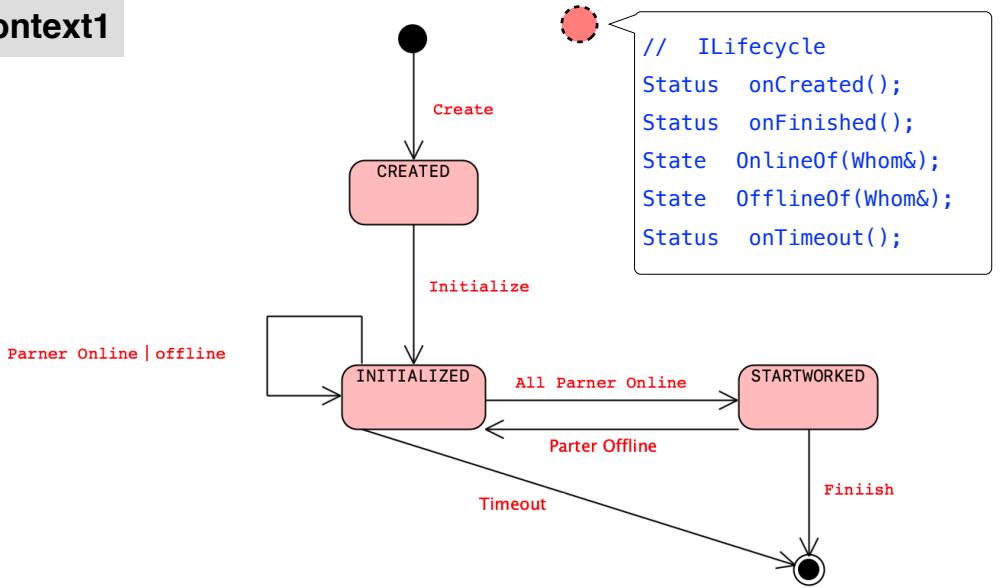
多范式融合C++软件设计



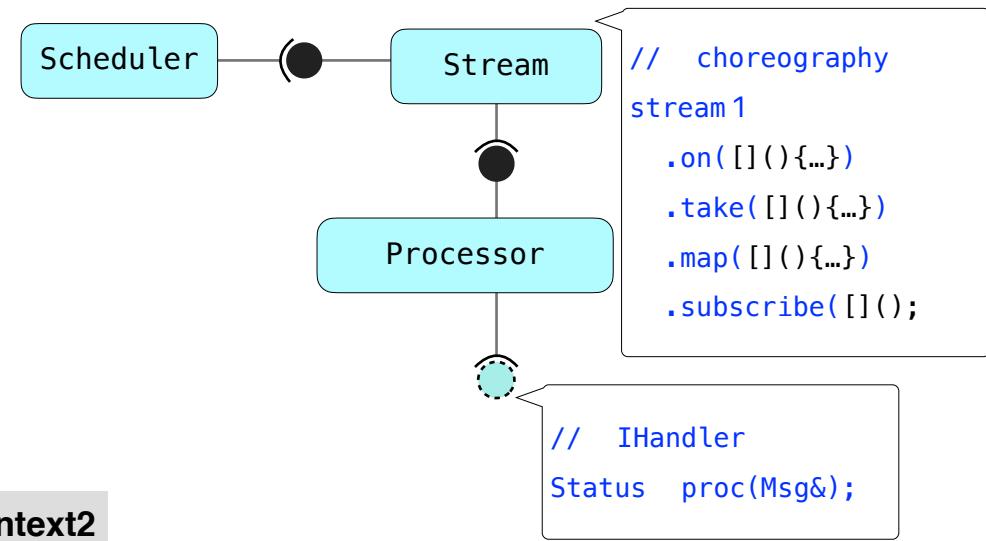
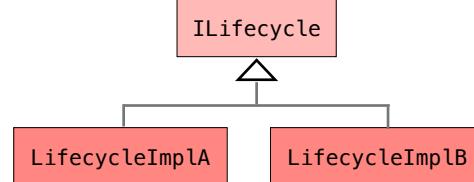
- 不做某单一范式的拥趸，分场景灵活选择合适的范式解决恰当的问题；
- 从领域驱动设计的角度，按照模型一致性，将不同主范式的设计划分为不同的域或层内；
- 根据不同域的范式特点，选择合适的C++特性予以支撑，不要滥用语法特性；
- **DCI架构**及其C++下的实现模式，是处理多范式融合设计的一种有效手段；

DCI设计过程

Context1

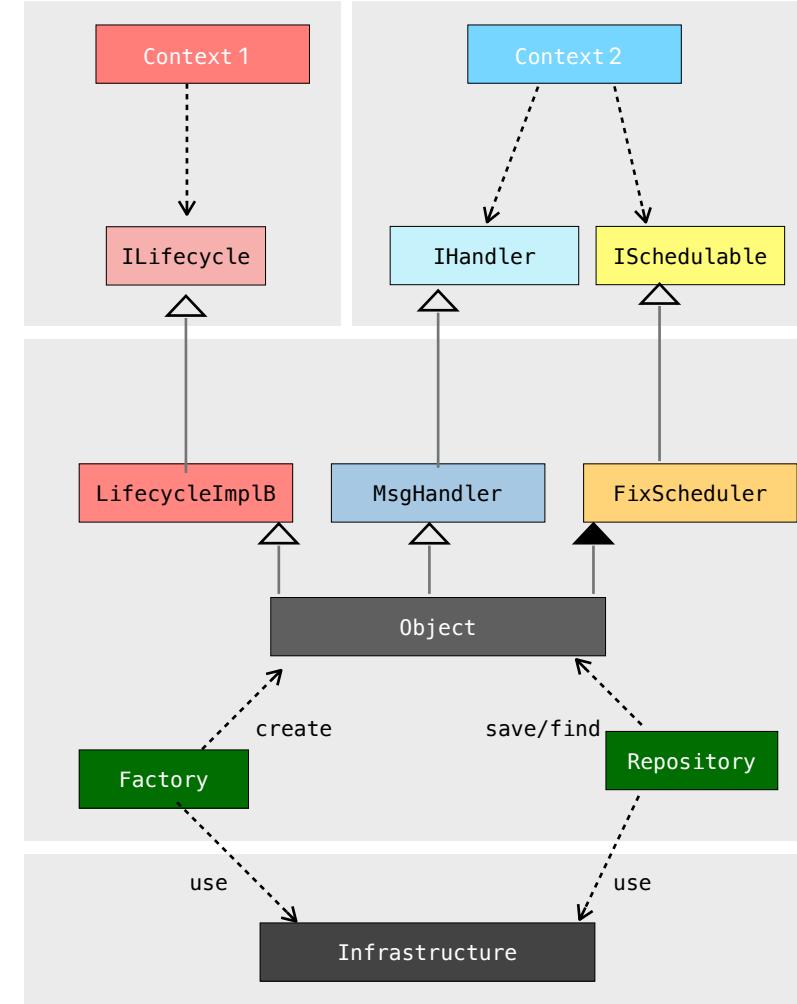


Role & Implementations



Role & Implementations

Combination



其它参考

- 2016全球C++及系统及软件大会《用组合式设计灵活高可靠的C++系统软件》
<http://boolan.com/lecture/1000001354>
- 2019全球C++软件技术大会《现代C++在在大型嵌入式系统中的应用和实践》；
<http://cpp-summit.org/speaker/235?uid=c1016>
- 2019全球C++软件技术大会《C++领域驱动设计应用与实践》
<http://cpp-summit.org/speaker/505?uid=c1016>
- 2020全球C++软件技术大会《面向领域模型的Modern C++实现模式》；
<http://cpp-summit.org/speaker/505?uid=c1016>

谢谢

尉刚强

软件技术咨询师



长期领导和赋能团队在各领域中实现自动化构建与测试、持续集成与部署、智能运维等各方面的能力。曾先后在嵌入式系统、4G/5G无线通信、大数据、人工智能对话平台、SAAS服务等领域中从事核心系统架构设计，领域建模和性能优化。作为全栈工程师，熟悉C/C++、Java、Scala、NodeJS，Ruby和Python等各种前后端技术和语言，并对不同语言设计与建模，构建差异、依赖管理设计等技术有深入研究。

主办方：

Boolan
高端IT咨询与教育平台

CPP-Summit 2020

尉刚强

软件技术咨询顾问

C/C++ 之 构建设计与重构

议程

1

C/C++构建介绍

2

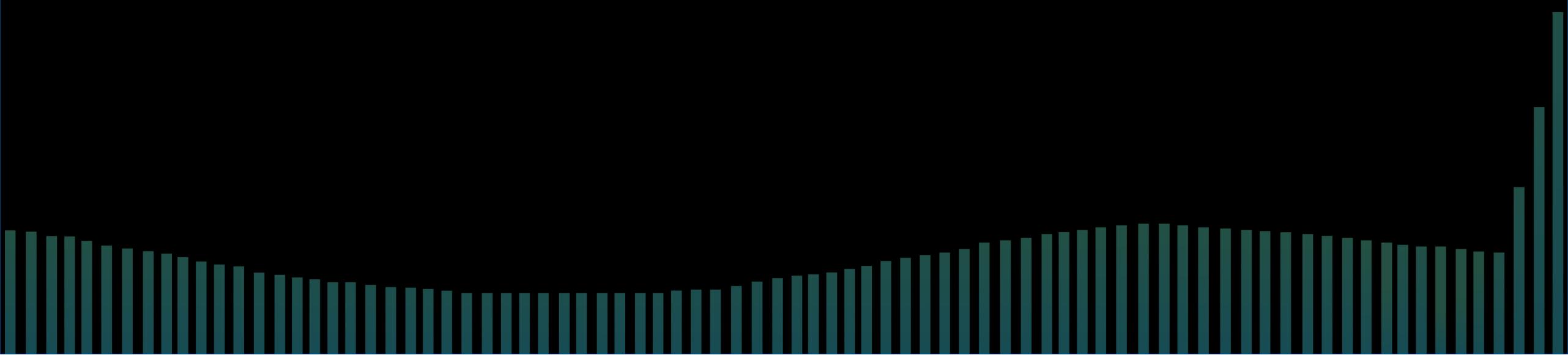
C/C++构建设计

3

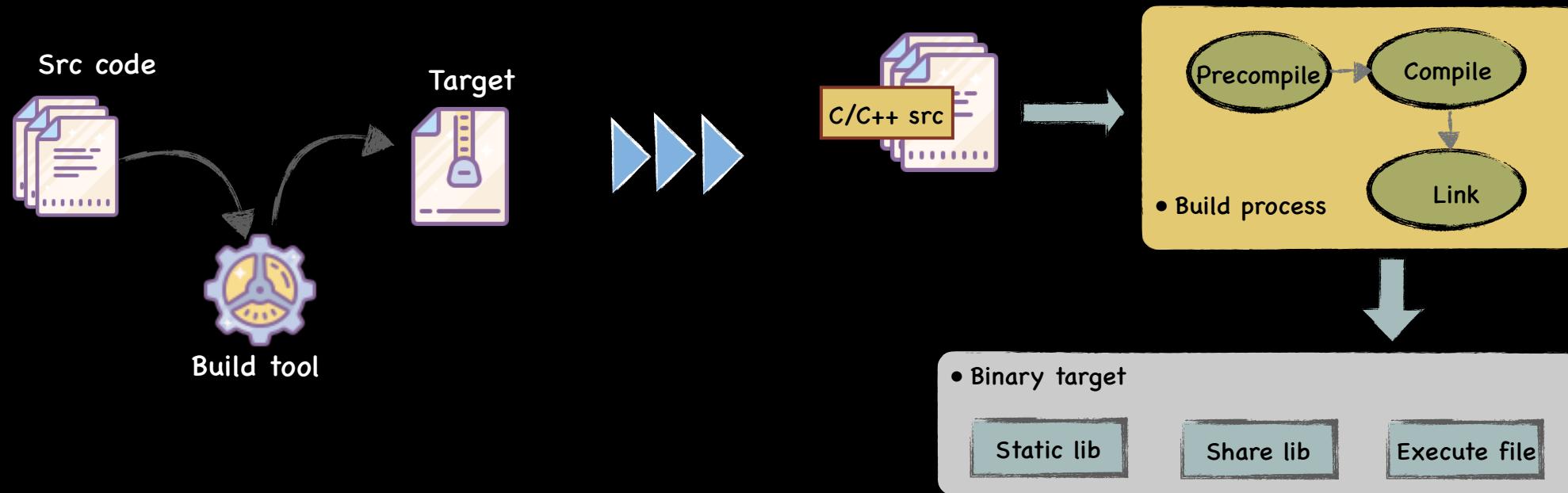
C/C++构建重构

01

C/C++构建介绍



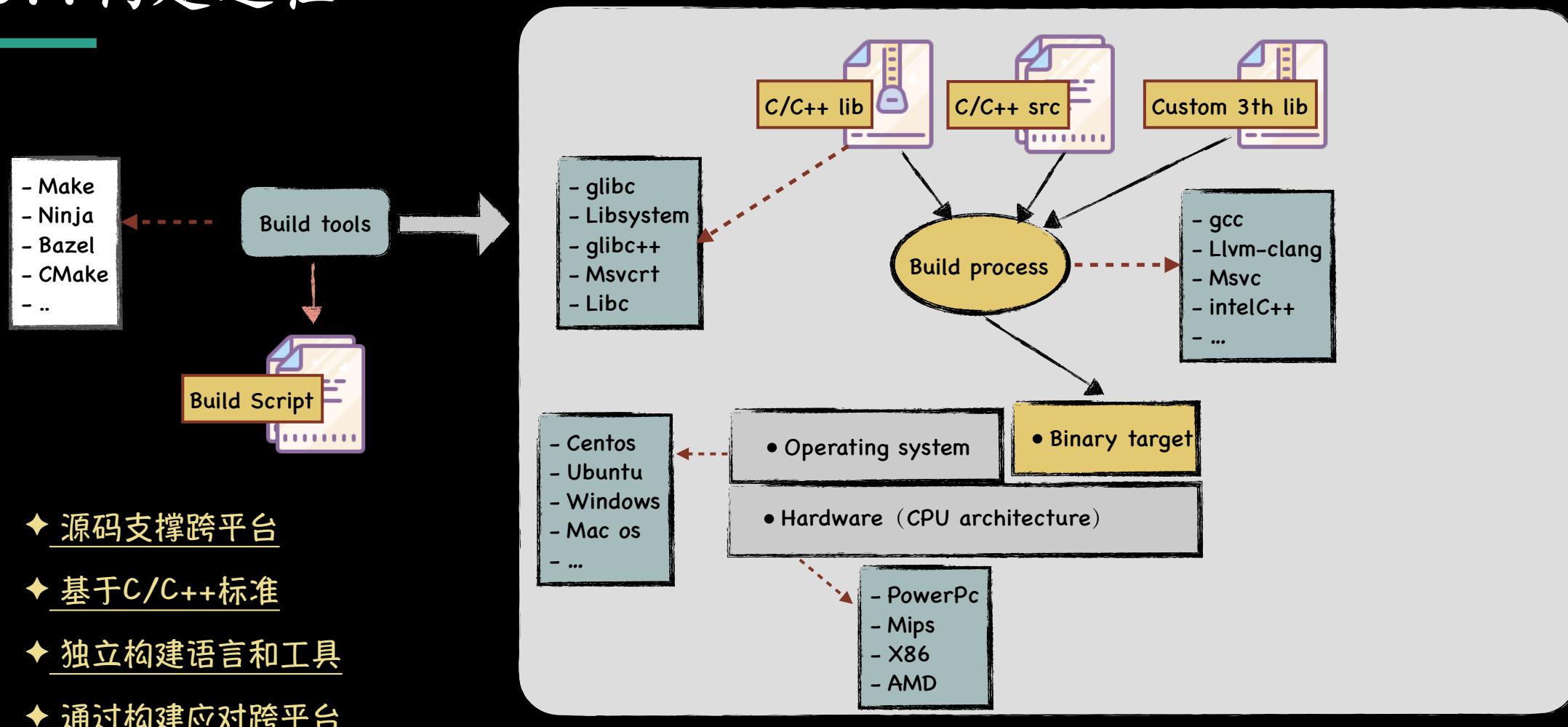
C/C++构建是什么？



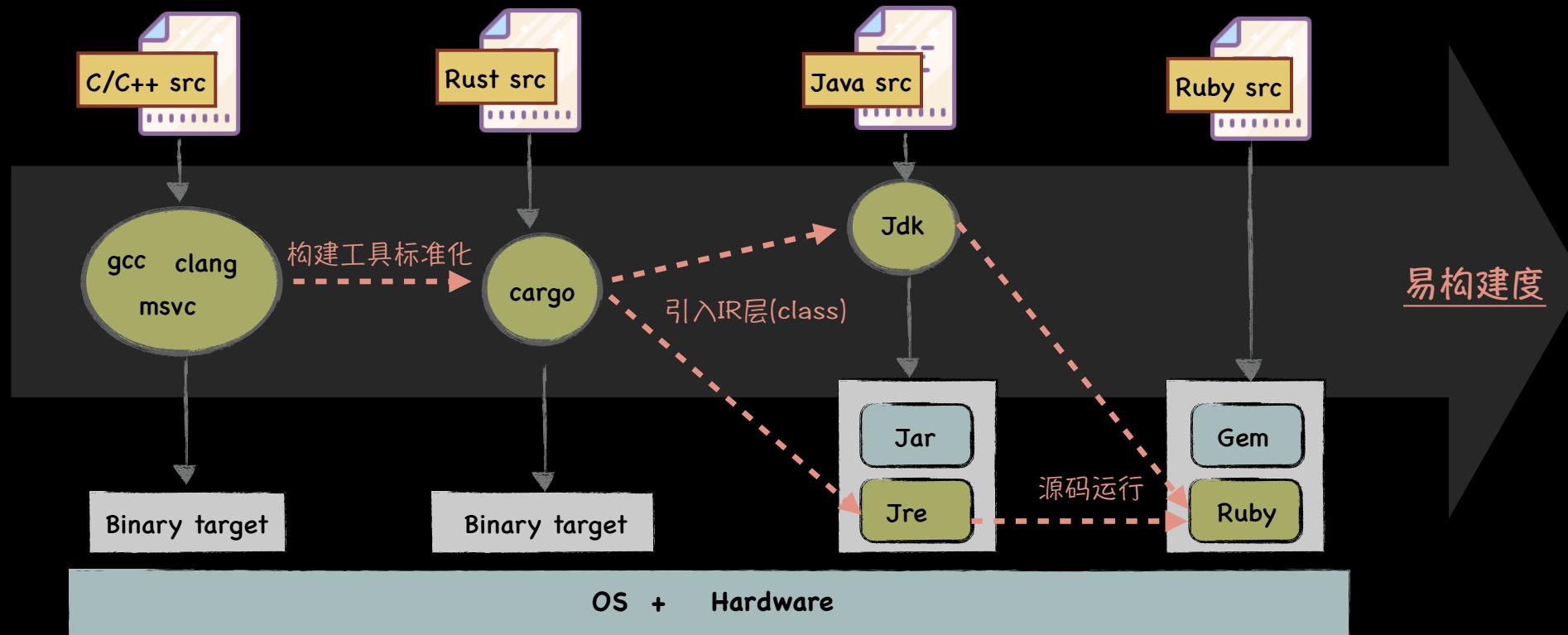
• 构建：源代码到目标的加工过程

• C/C++构建

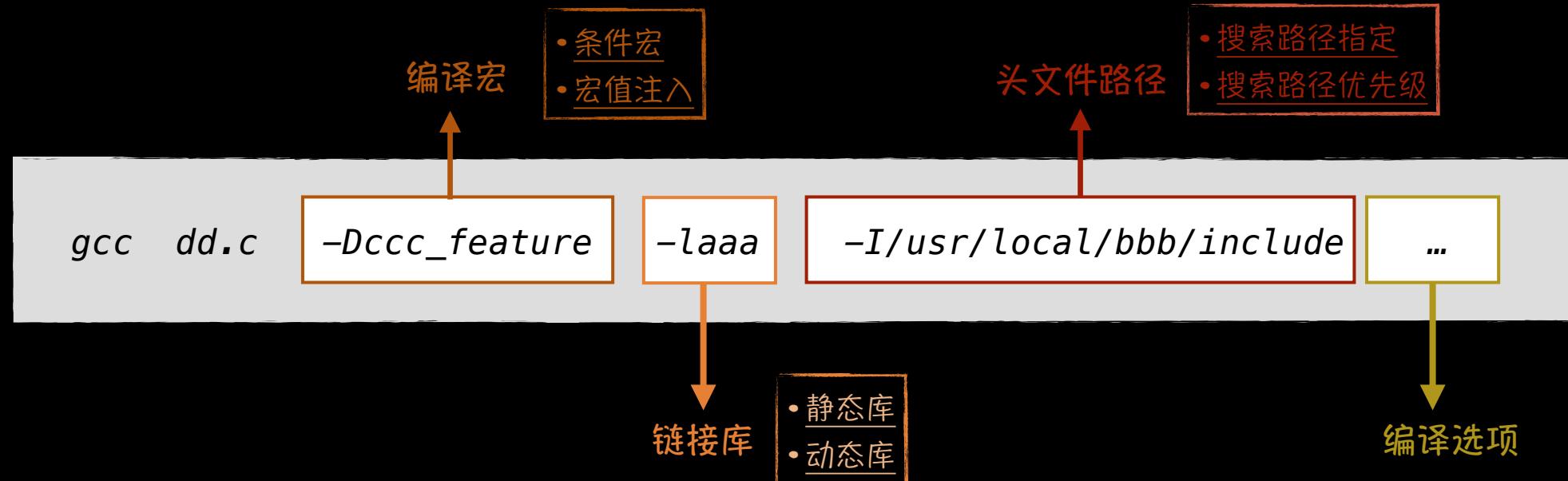
C/C++构建过程



C/C++与其他语言构建对比

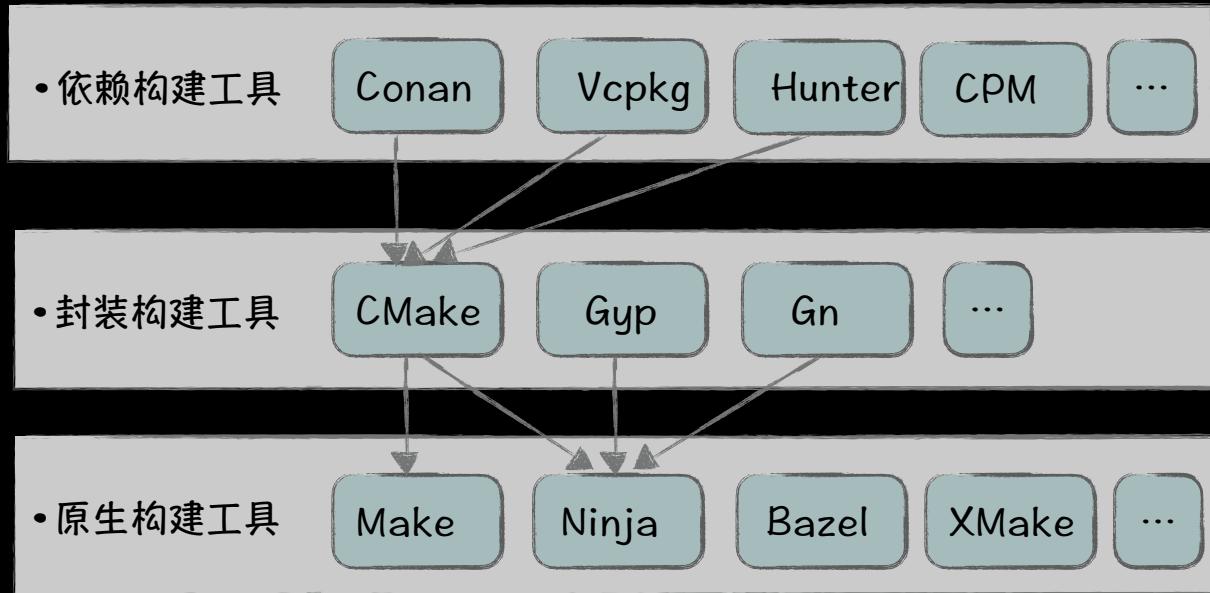


C/C++语言构建复杂性



◆ 软件设计、构建设计、依赖管理 耦合

C/C++的构建语言工具

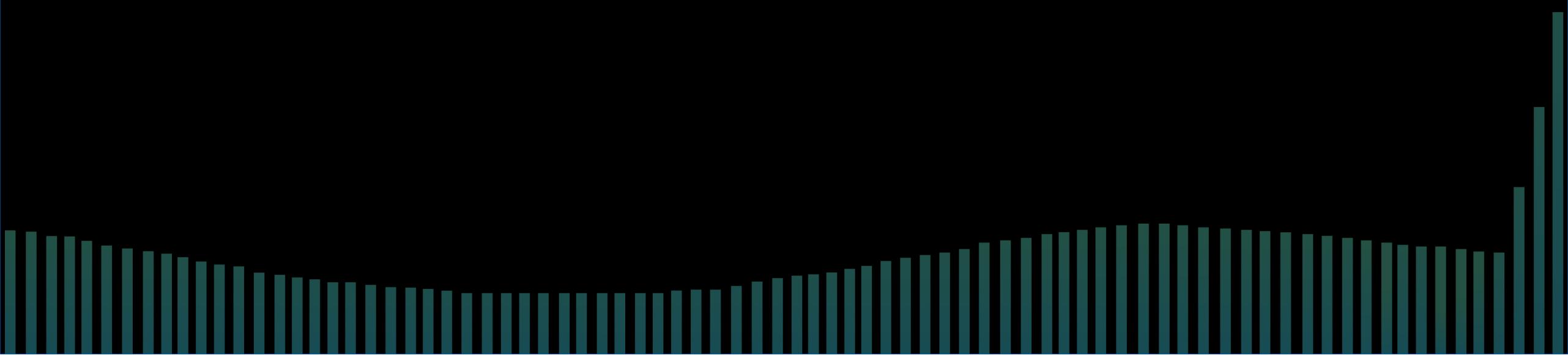


特点：

- 更关注性能
- 抽象级别更高
- 构建描述更简洁
- 可读性更强

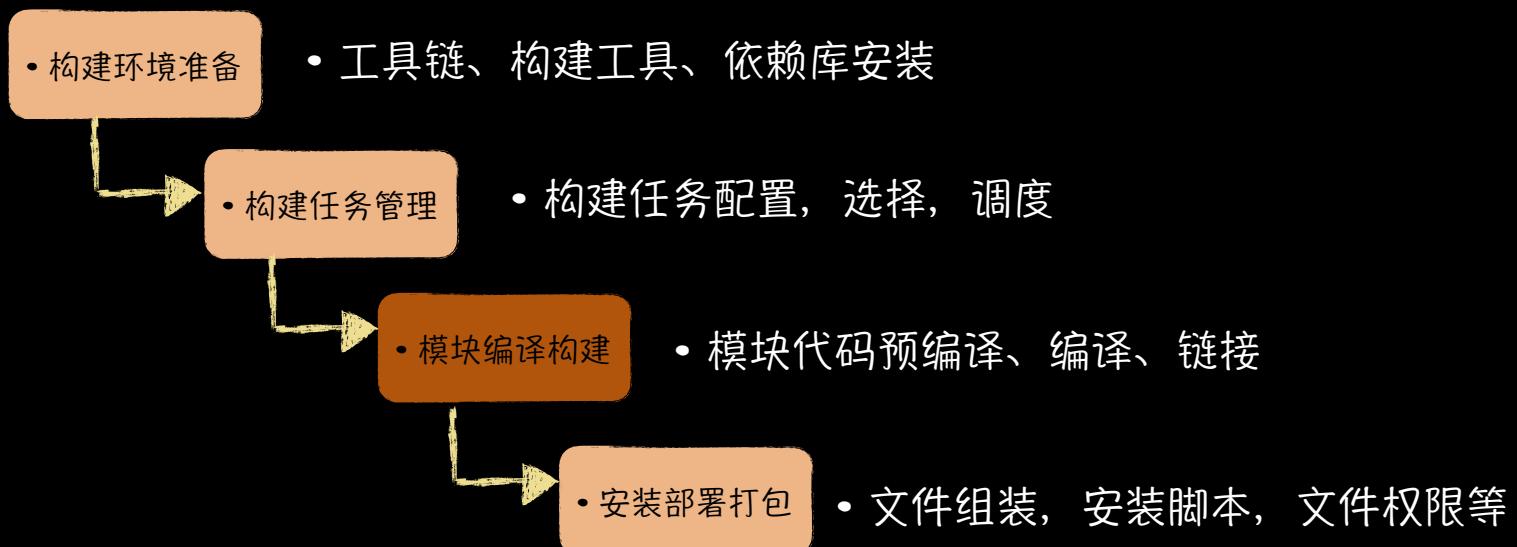
02

C/C++构建设计



C/C++构建设计

▶ 大型嵌入式系统云构建



关注点：

- 构建可重复
- 构建性能高
- 构建代码化
- 构建可视化

C/C++构建设计

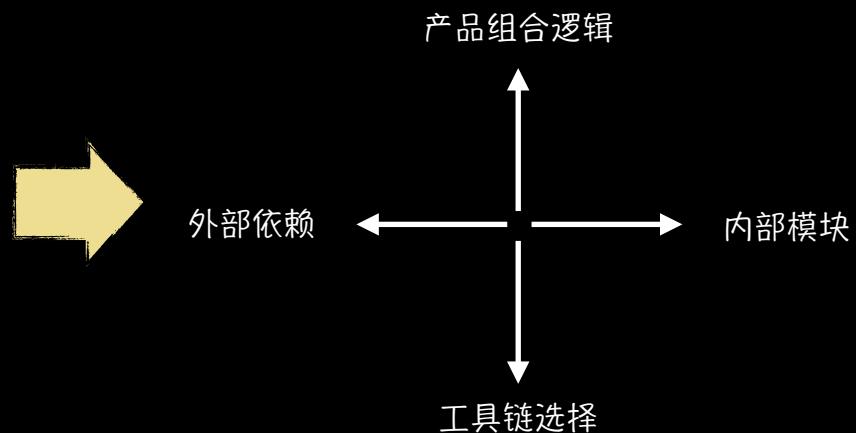
► 模块编译构建



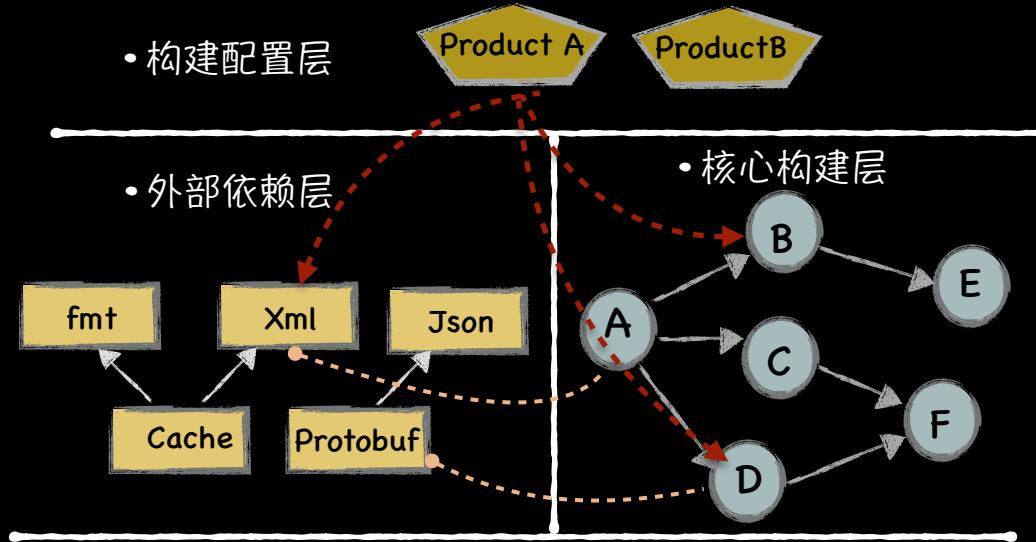
问题：

- 构建修改难度大
- 模块独立构建测试难
- 构建性能差

- 构建核心领域变化方向



C/C++构建设计

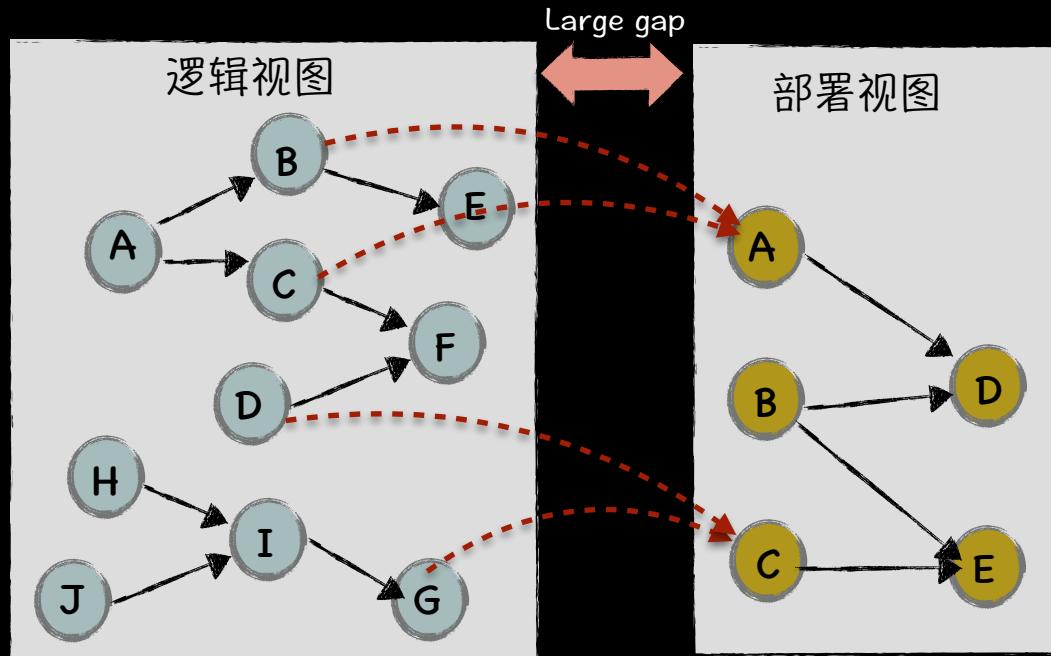


设计原则：

- 分离不同变化方向
- 不同层构建脚本独立
- 通过组合完成系统构建

C/C++构建设计

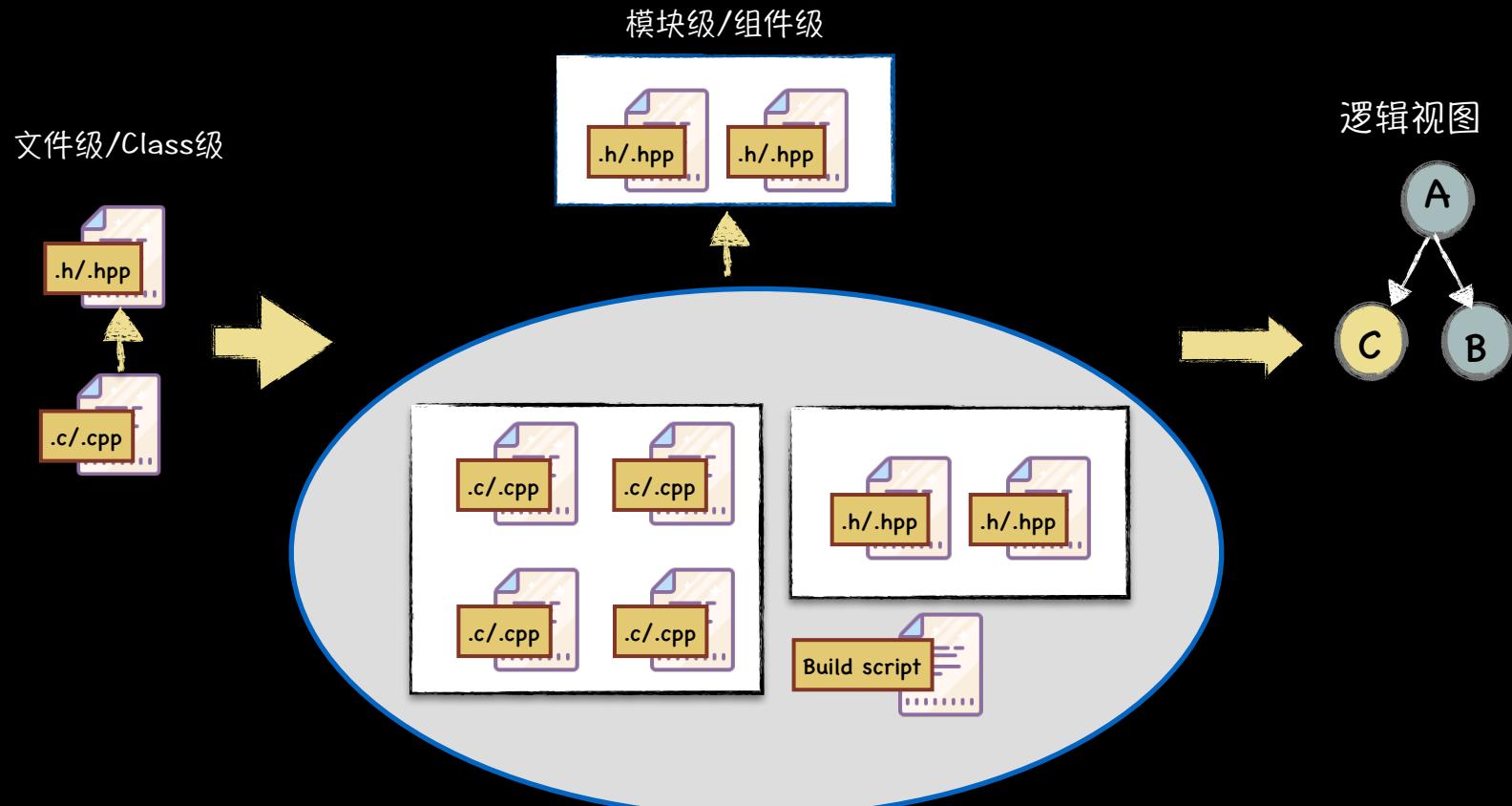
▶ 核心构建层设计



问题：

- 开发远离构建
- 构建粒度太粗
- 构建脚本复杂

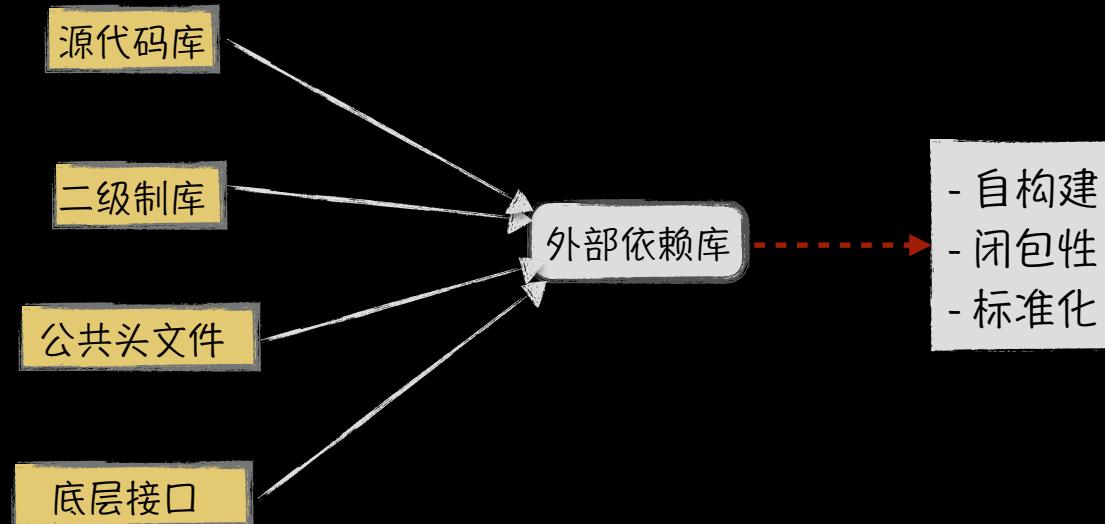
C/C++构建设计



- 构建支撑更大粒度的封装
- 构建与物理设计保持一致
- 构建节点对象化

C/C++构建设计

► 外部依赖层



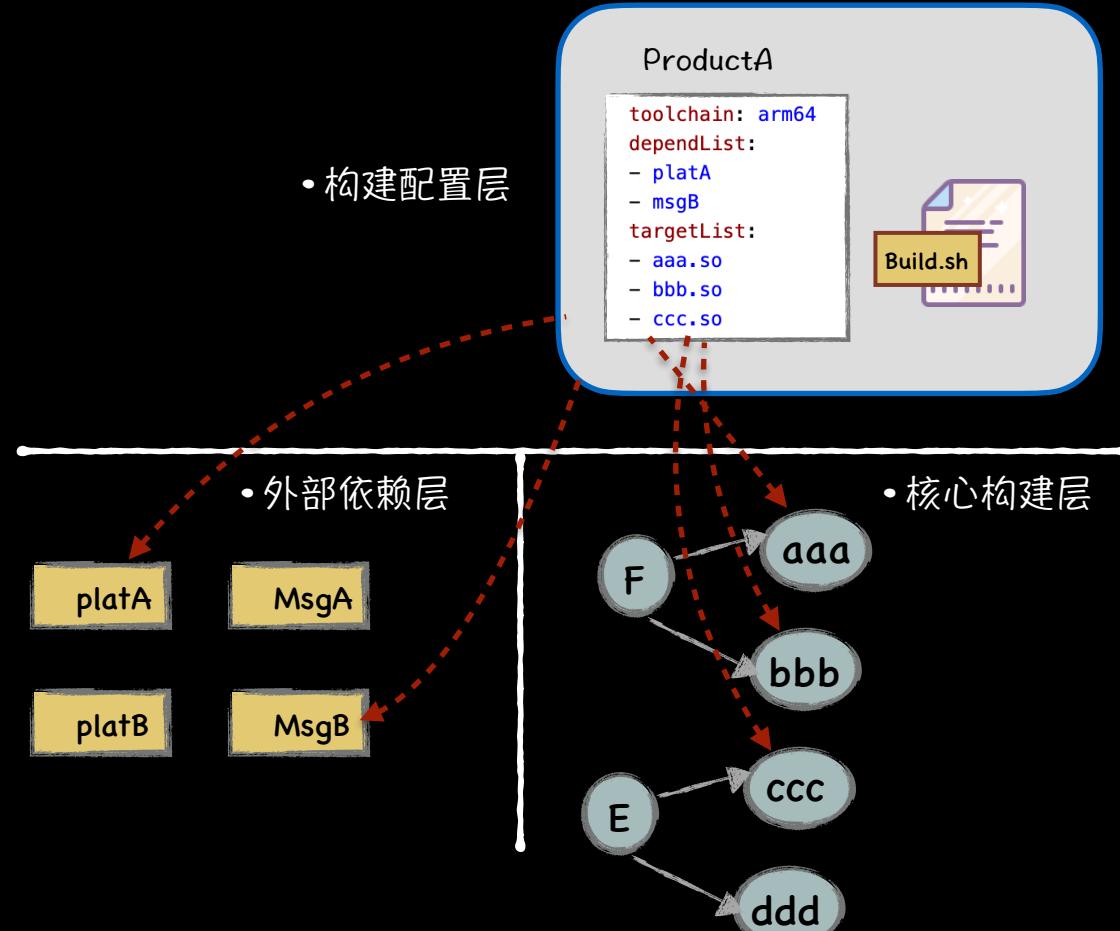
外部依赖界定：

- 非构建系统内提供的代码元素

挑战：

- 头文件构建选择
- 精确依赖控制
- 编译宏选择

C/C++构建设计之案例

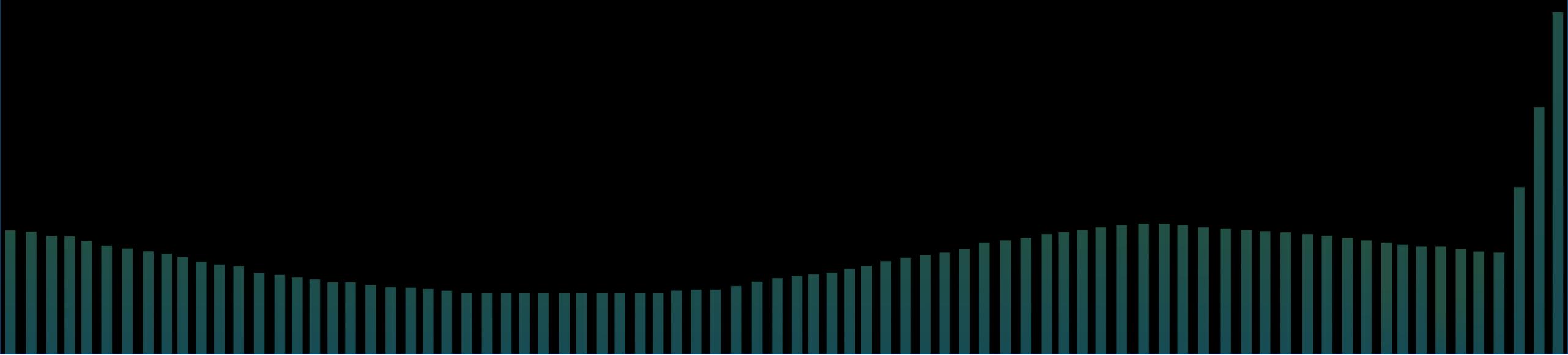


优点：

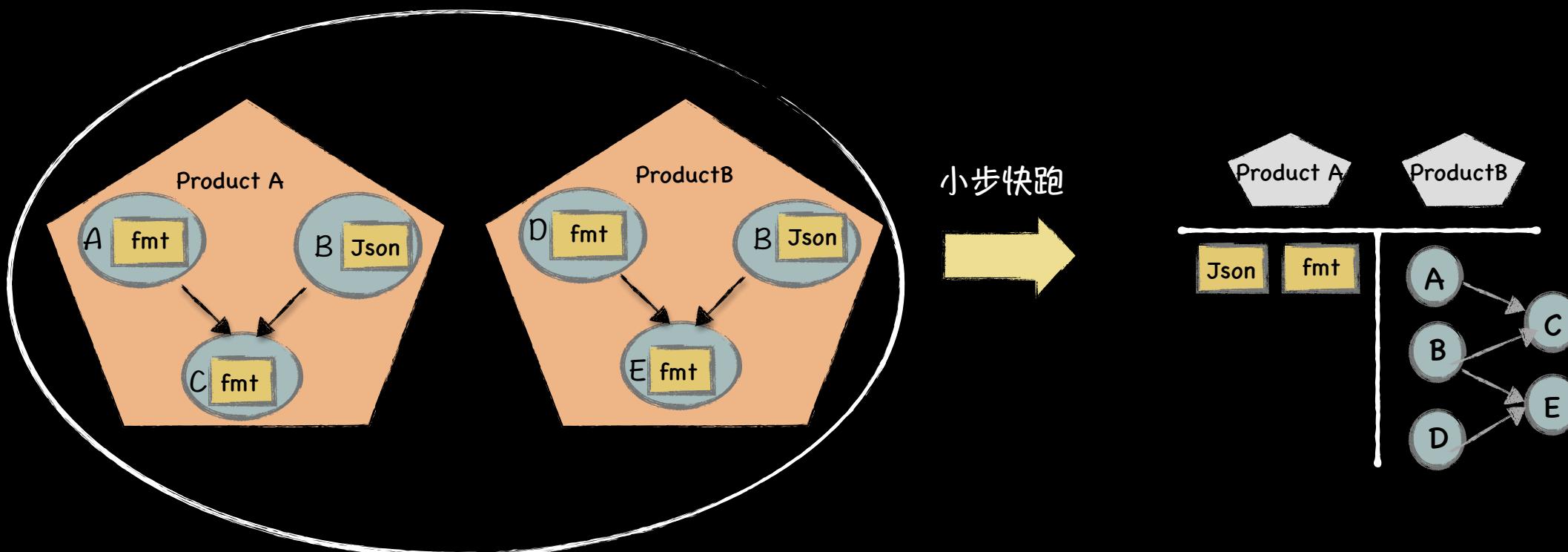
- 支持多款CPU硬件类型
- 产品灵活选择外部依赖
- 产品灵活选择构建组件

03

C/C++构建重构



C/C++构建重构



C/C++构建重构

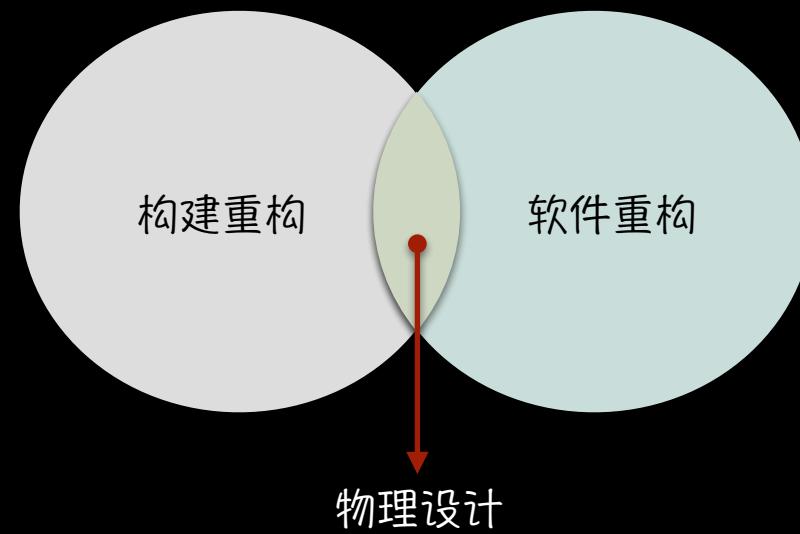
构建重构： 对构建脚本的调整， 目的在不改变软件构建目标行为前提下， 提升构建脚本的可读性，
降低其构建修改成本， 提升构建效率。

目标：

1. 正确完成构建目标功能；
2. 提高构建效率；
3. 消除构建脚本重复；
4. 提高构建脚本可理解性；
5. 减少冗余

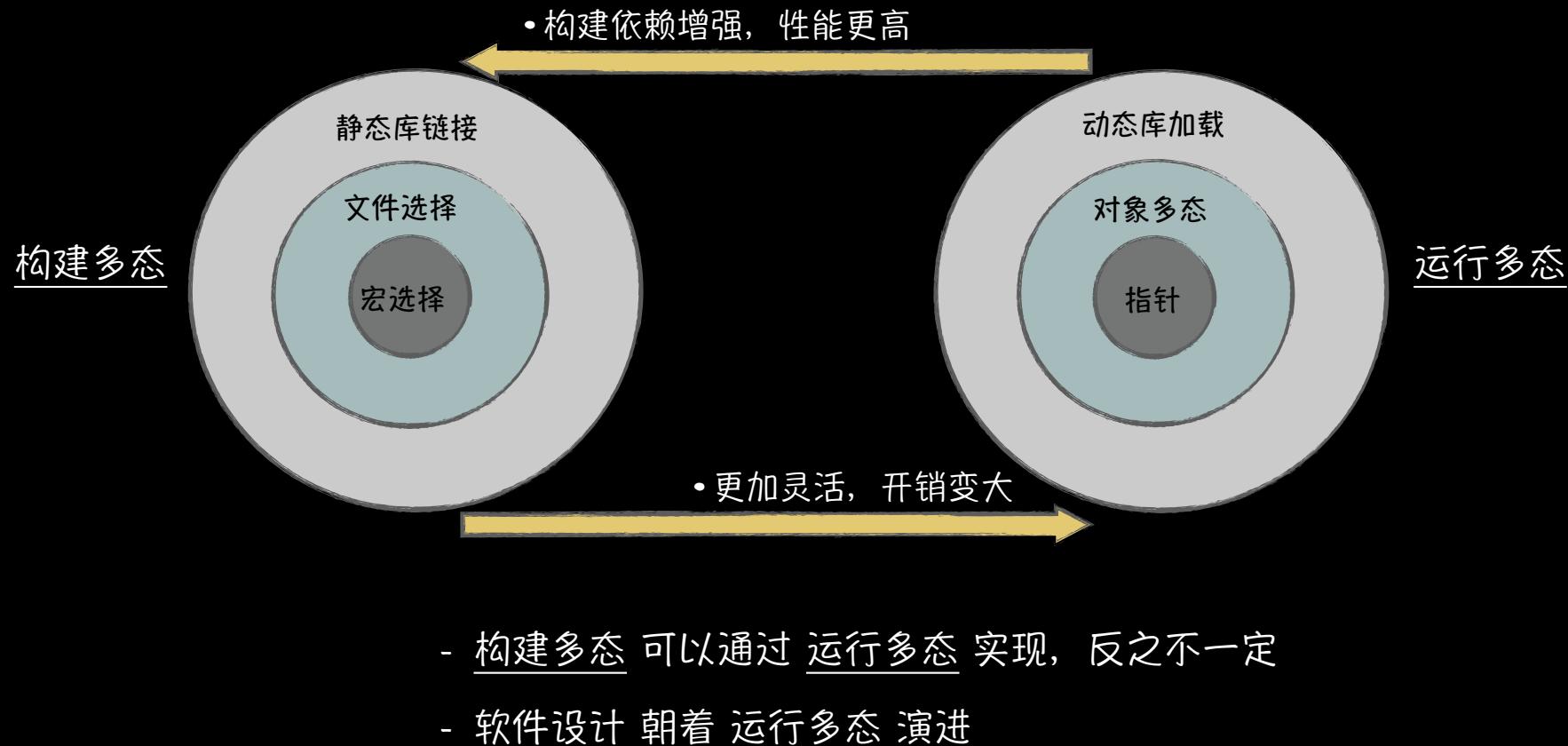
C/C++构建重构

- 构建脚本元素的调整
- 没有构建设计指导
- 重构社区经验很少



- 软件代码元素的调整
- 有软件设计指导
- 重构方法成熟，有工具支撑

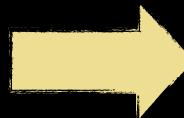
C/C++构建重构



C/C++构建重构安全网

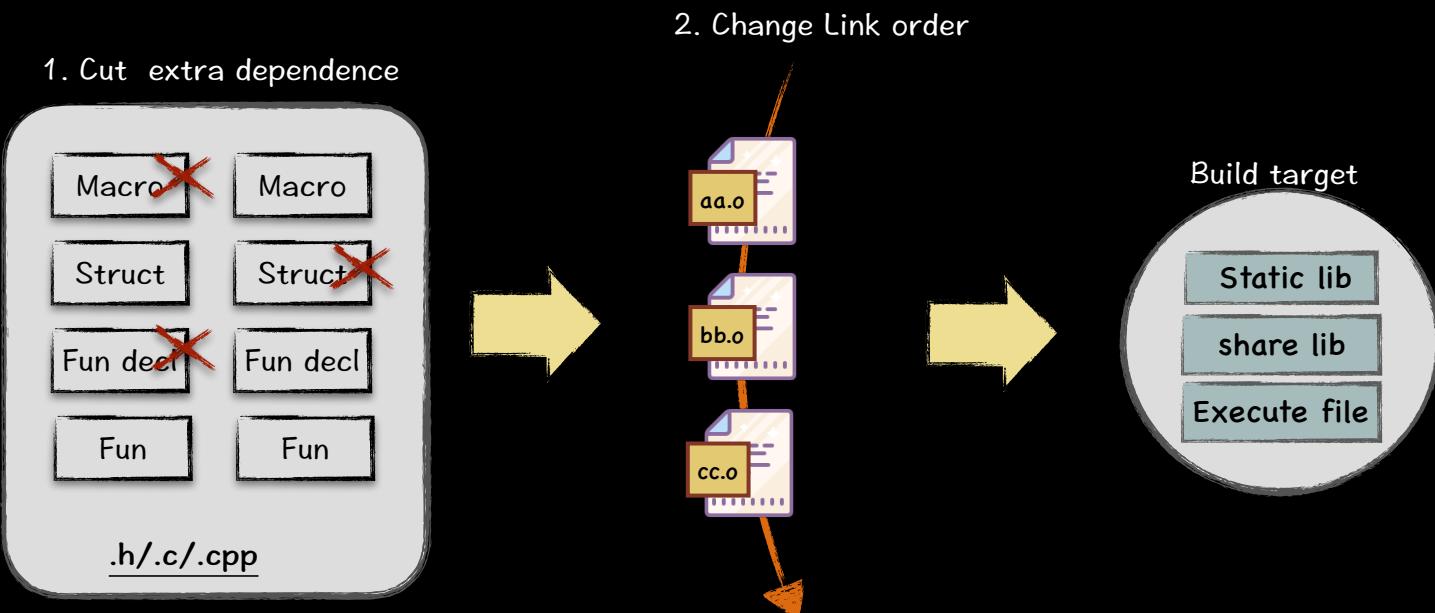
可以用自动化测试来验证吗？

- 没有自动化测试用例
- 自动化用例不全
- 自动测试执行时间太长



构筑仅用于构建重构的安全网！

C/C++构建重构安全网

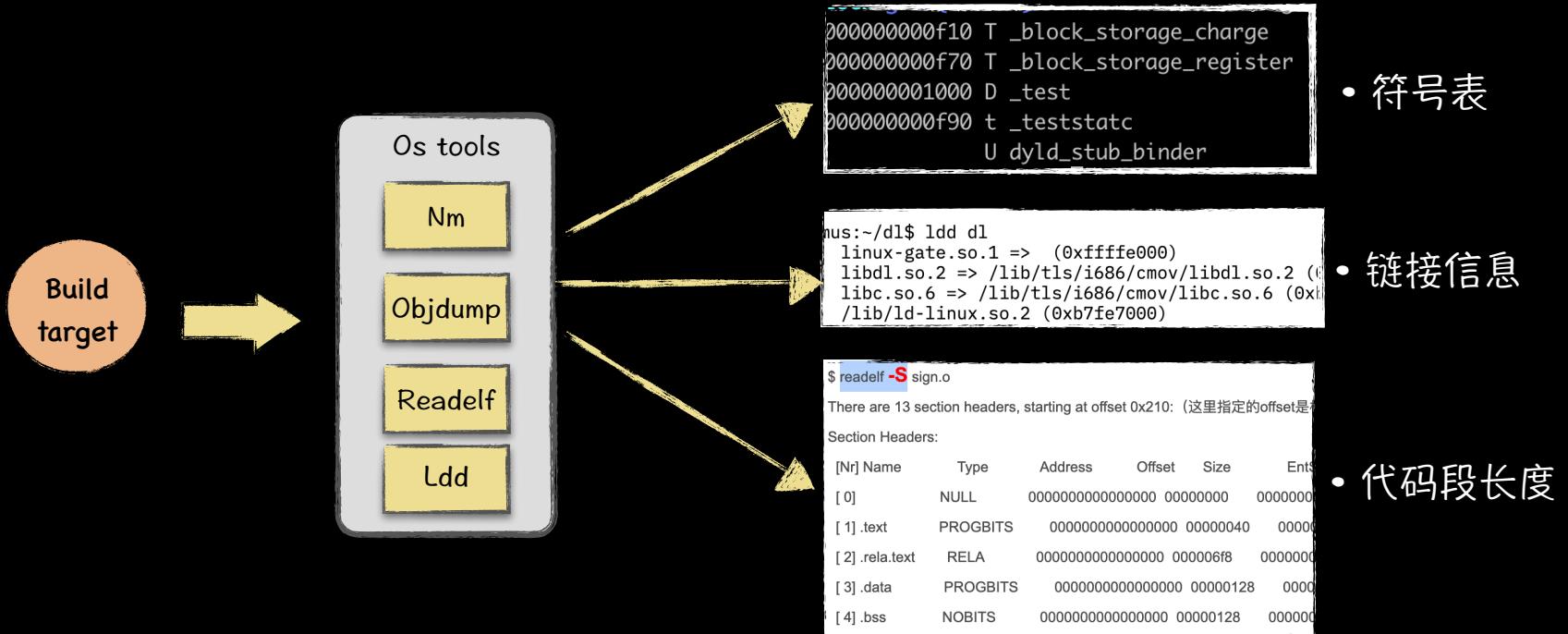


现象：

- 二进制大小会改变
- 二级制内容会改变
- 不同类型文件格式不同

◆ 怎样保证构建生成的二级制行为不变呢？

C/C++构建重构安全网



• 符号表

• 链接信息

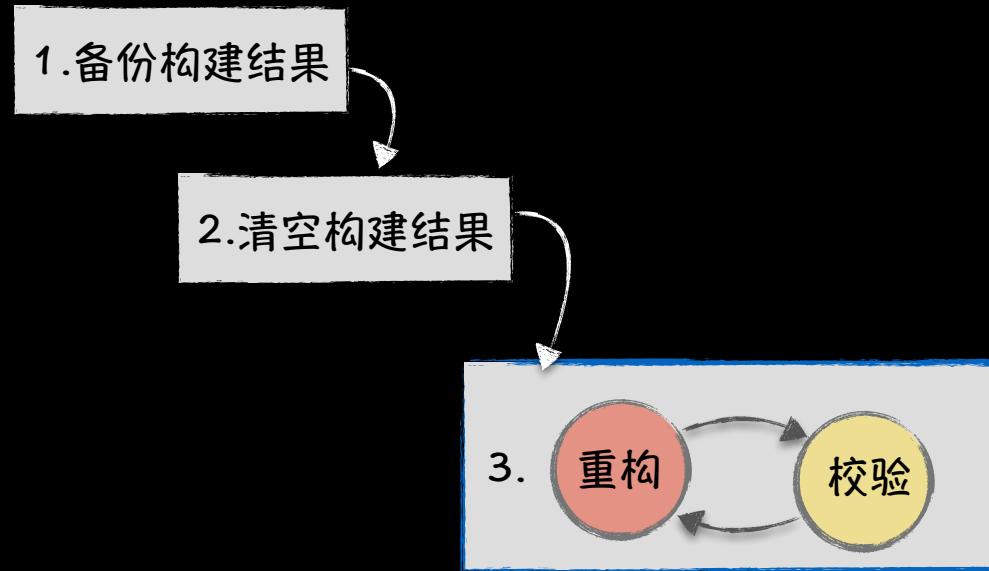
• 代码段长度

校验内容：

- 符号表和类型
- 动态库链接信息
- 代码段数据段长度

◆ 构建测试安全网能确保100%的安全吗？

C/C++构建重构安全网



校验内容：

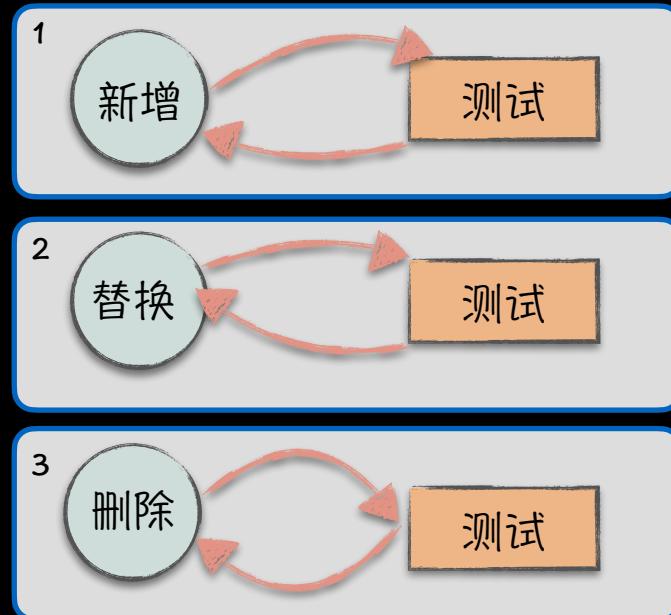
- 二进制文件对比
- 符号表对比
- 安装位置校验
- 工具链遍历构建
- 产品宏定义遍历构建
- 构建时间校验
- ...

◆ 基于shell构筑测试安全网，自动化运行

C/C++构建重构坏味道

- 构建脚本代码重复
 - 构建脚本与模块位置不一致
 - 构建脚本中包含大量绝对路径
 - 过长的构建脚本
 - 过多依赖环境变量
 - 过多的逻辑判断
 - ...
- 过多的全局变量
 - 变量作用域太长
 - 依赖冗余的动态库
 - 过长头文件 (调整物理设计)
 - 头文件位置不合理 (调整物理设计)
 - 冗余的编译宏
 - ...

C/C++构建重构手法



重构项：

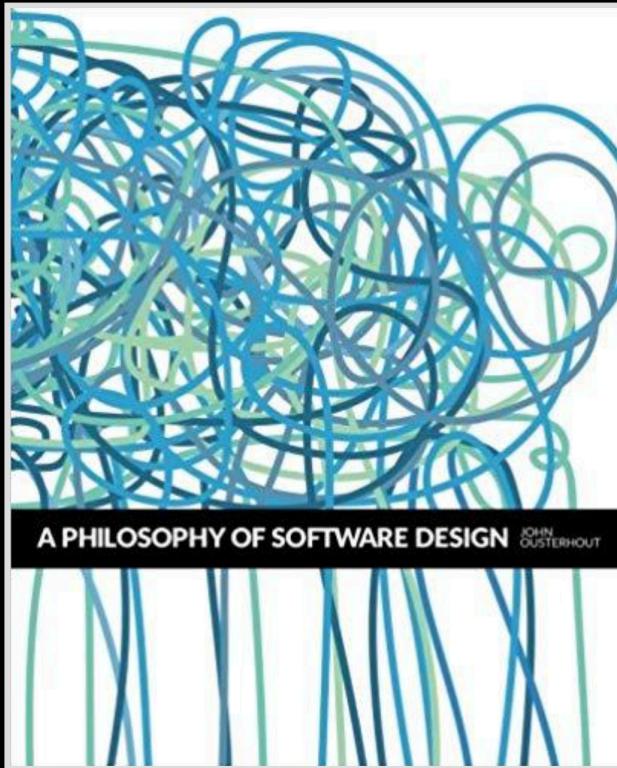
- 提取依赖库
- 提取静态库
- 提取构建函数
- 提取构建变量
- 删除全局变量
- 头文件拆分
- 头文件移动
- ...

C/C++构建重构



- ◆ 调整头文件路径依赖顺序会影响编译结果吗？
- ◆ 调整源文件的链接顺序会影响构建的二级制文件大小吗？
- ◆ 调整构建脚本的位置会影响构建dbg文件大小吗？

总结



软件设计应该简单、避免复杂

- 软件设计的哲学

谢谢

吴锐

腾讯TEG云架构平台部高级工程师



腾讯高级工程师，从事CDN后台基础架构和优化。专注于开发高性能Web服务器的架构和CDN基础组件的开发与设计。参与过CDN服务器开发，CDN内核优化等项目。

主办方：

Boolan
高端IT咨询与教育平台

C++ Summit 2020

吴锐
腾讯高级工程师

C++高性能大规模服务器 开发实践

01

Lego简介

02

传统Web框架

03

Lego架构实现

04

未来展望

00. Who am I

姓名：吴锐

英文名： royrwu

经历：加入腾讯后一直从事与CDN架构和运营相关的工作。参与过高性能服务器，CDN内核和海量运营相关的开发工作。

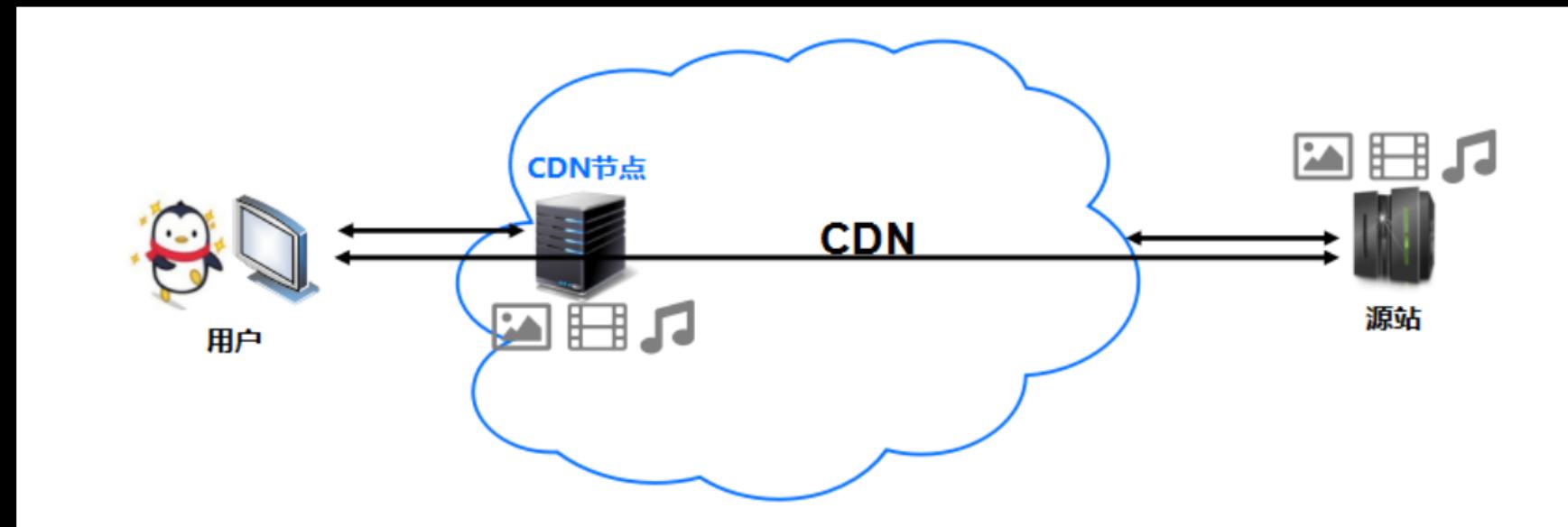
目前是腾讯CDN服务器开发技术负责人。

长期专注于CDN相关服务器，内核和网络等相关技术的研发和实践。

01. Lego简介 — CDN

CDN, Content Delivery Network, 全称内容分发网络。就是网络世界的快递公司，有着全国，全球的快递网络，网点。将快递用最快的速度从“卖家”（源站）发送到“买家”（客户端）手上。区别在于一份快递会重复发送的全国不同的地方，并且可以从“网点”（边缘节点）直接发货。

根据运营商，区域，负载，链路情况等因素提供最优节点给客户端就近访问。



01. Lego简介 – CDN

100+^{Tbps}

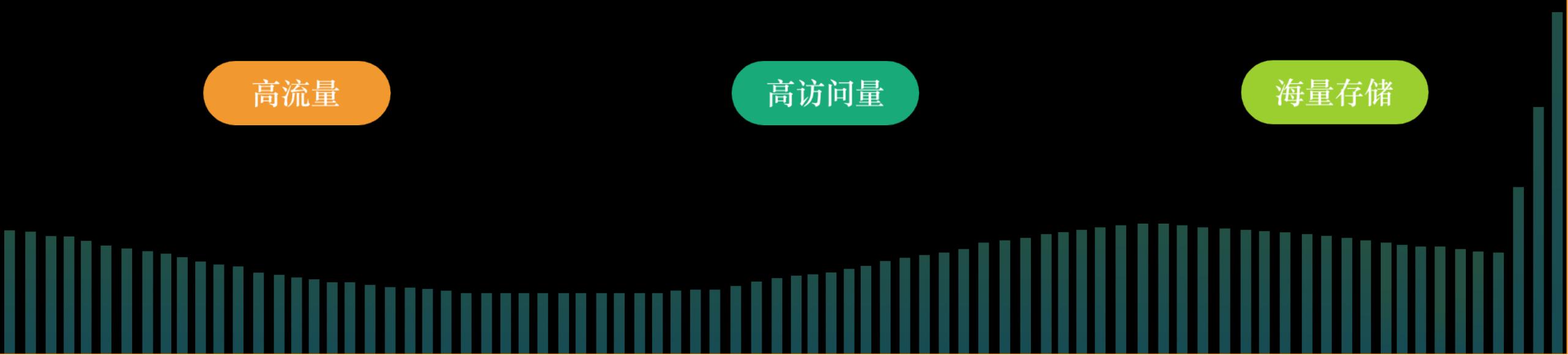
高流量

1千万+^{QPS}

高访问量

1 亿

海量存储



静态加速

电商、门户、APP中静态的图片、页面资源访问加速



流媒体点播加速

视频网站中，流媒体HTTP下行加速



腾讯CDN
支持产品

APP分发、游戏升级包、手机固件升级等大内容下载

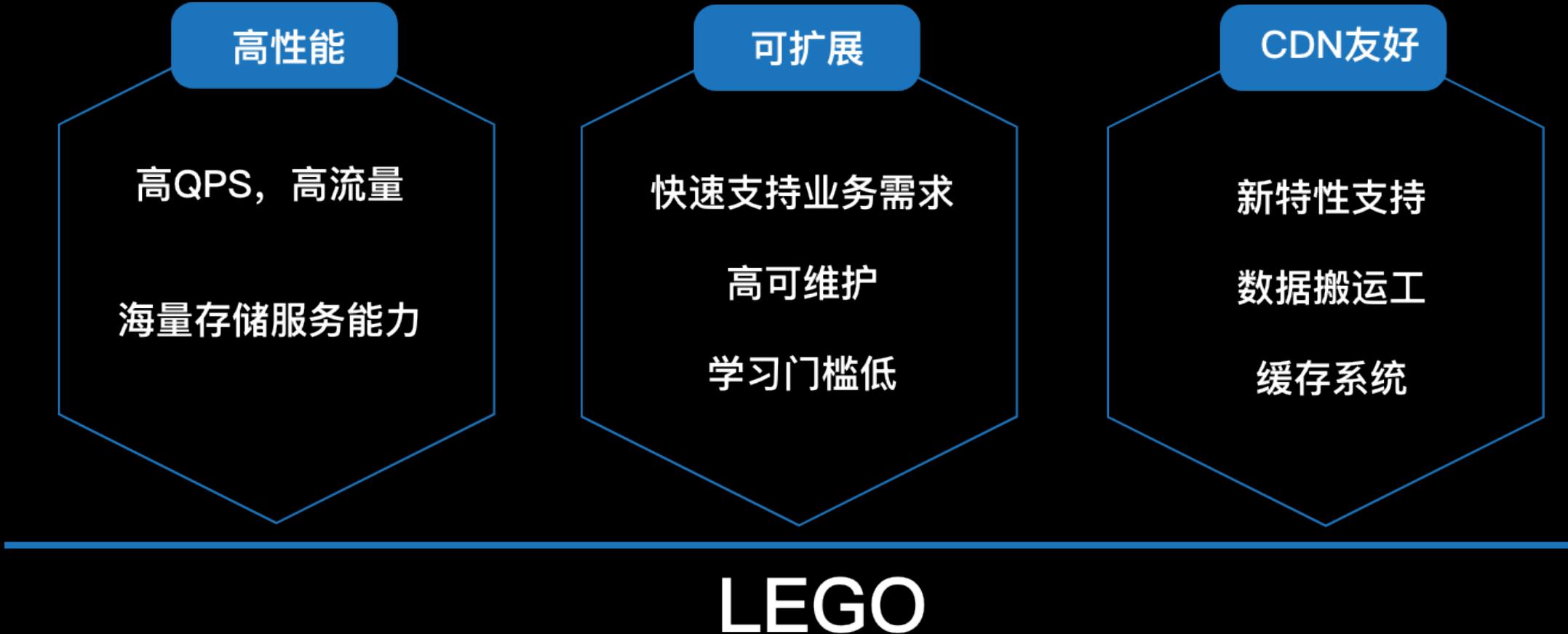


流媒体直播加速

直播、互动直播等场景中，下行流分发加速



01. Lego简介 — CDN服务器



02. 传统Web框架



异步回调

框架内部提供请求不同阶段的Hook，通过不同的Hook来实现功能。

需要对框架十分了解，调试比较复杂



Coroutine

基于协程来编程，应用程序通过协程库函数来驱动服务的运行，函数本身也要针对不同的事件调用对应的处理函数。

存在协程切换开销，上下文存储内存开销



Continuation

基于Continuation概念进行编程，具体事件被触发时调用Continuation，CPS编程模式的前身。

Continuation本身含有锁，并且Continuation与框架紧耦合。

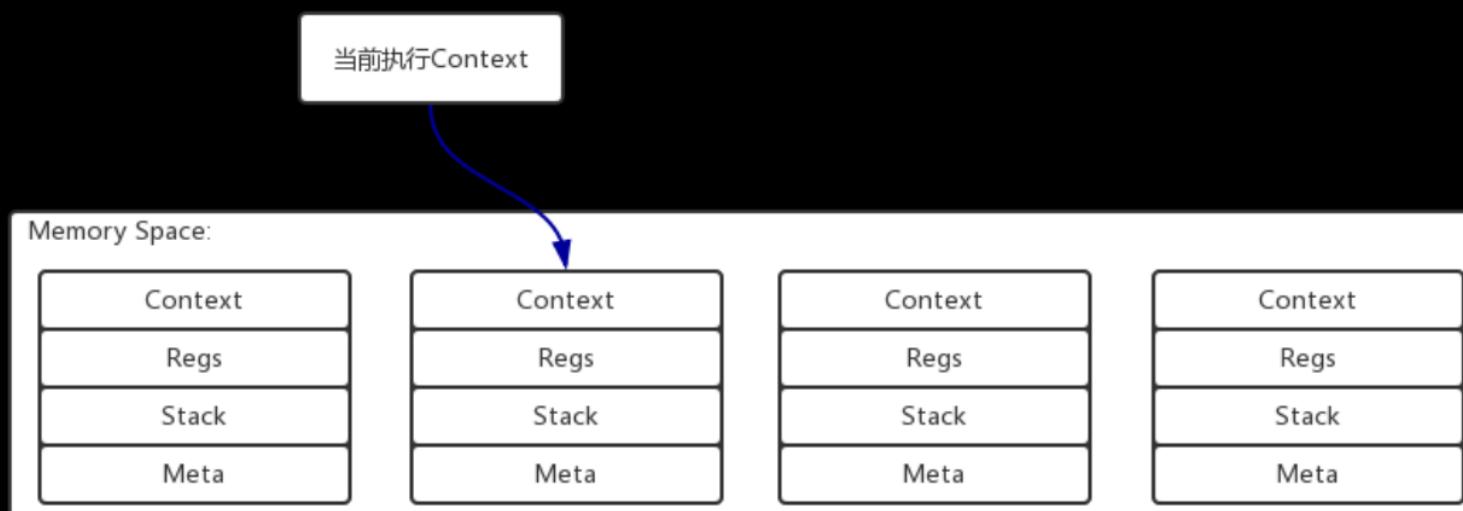
02. 传统Web框架

- 异步回调可维护性和可扩展性不佳：

- Nginx将一个请求区分为11个阶段，哪个阶段实现逻辑最为合适？
- Nginx框架通过设置不同的event handler将事件串联，代码逻辑分散在各个event handler中，如何管理代码？

- Coroutine栈空间分配管理复杂：

- 协程栈空间大小设置为多少合适？
- 协程空间的拷贝带来的性能开销有多少？
- 协程还是基于对不同事件的处理来实现业务逻辑，与异步回源的区别？



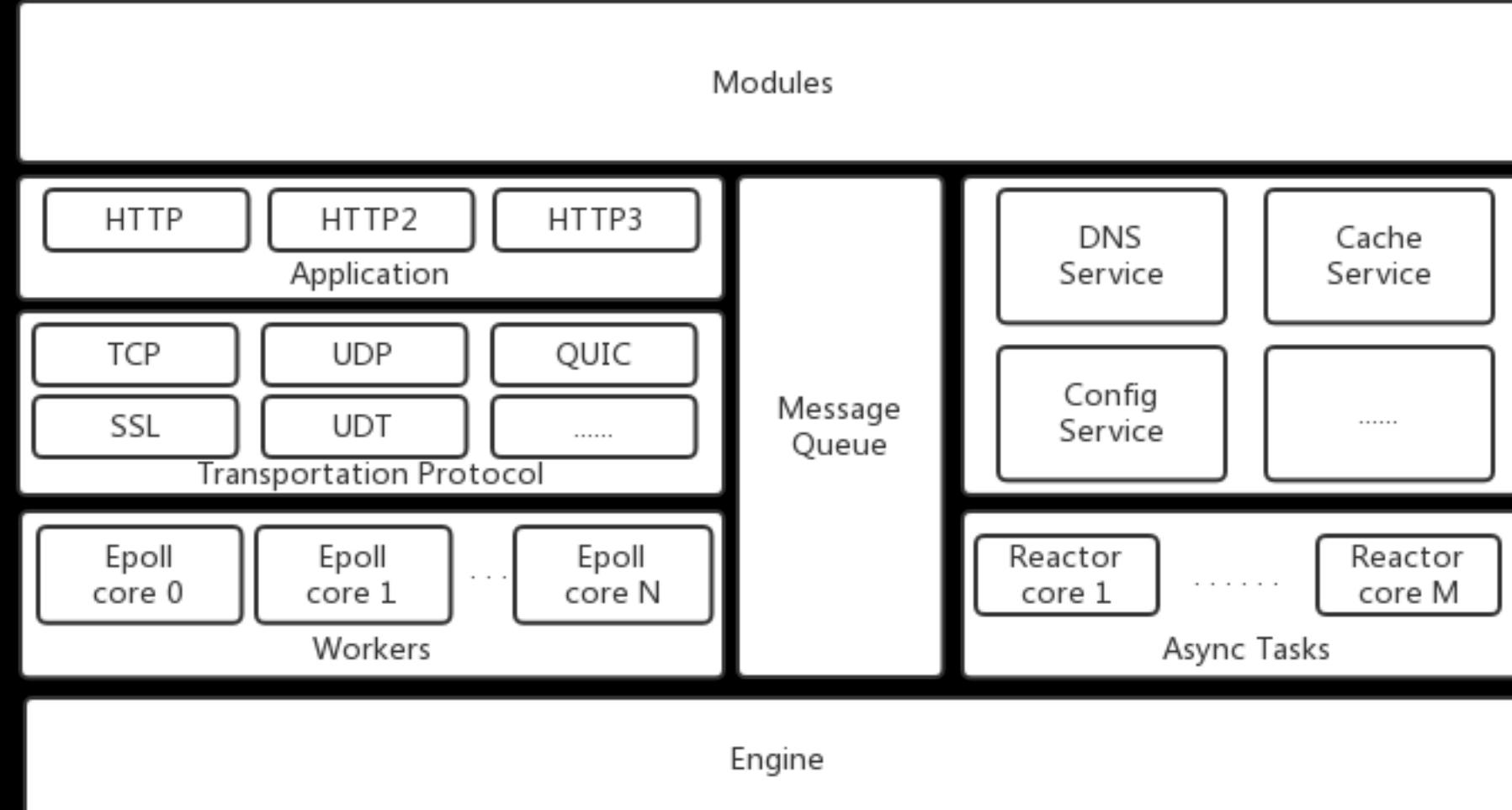
| NGINX 阶段枚举 |
|-------------------------------|
| NGX_HTTP_POST_READ_PHASE |
| NGX_HTTP_SERVER_REWRITE_PHASE |
| NGX_HTTP_POST_REWRITE_PHASE |
| NGX_HTTP_PREACCESS_PHASE |
| NGX_HTTP_ACCESS_PHASE |
| NGX_HTTP_POST_ACCESS_PHASE |
| NGX_HTTP_PRECONTENT_PHASE |
| NGX_HTTP_CONTENT_PHASE |
| NGX_HTTP_LOG_PHASE |

02. 传统Web框架

- Continuation-Passing Style整个开发流程基本串行执行
- Continuation基本没有性能开销
- ATS中实现的Continuation并非完美的Continuation
- C++11实现了Continuation关键技术Lambda，并引入了future的概念
- C++新的future extension丰富了future的语义

```
TS_INLINE int
handleEvent(int event = CONTINUATION_EVENT_NONE, void *data = nullptr)
{
    // If there is a lock, we must be holding it on entry
    ink_release_assert(!mutex || mutex->thread_holding == this_ethread());
    return (this->*handler)(event, data);
}
```

03. Lego架构 – 架构图



03. Lego架构 – 异步回调之痛

```
void HandleRequest(Request req)
{
    //do something...
    req.SetReadHandler(ReadRequestHandler);
    req.SetWriteHandler(ErrorWriteHandler);
}

void HandleUpstreamResponse(Request req)
{
    //do something...
    req.SetReadHandler(ReadUpstreamResponseHandler);
    req.SetWriteHandler(ErrorWriteHandler);
}

int ErrorWriteHandler(Request req)
{
    // Something went wrong. How did I get here?
}
```

当发生异常时，导致异常的元凶已经逃离现场。Debug过程变得十分困难，主要依赖于程序员的额外记录的信息与经验。

之前执行的是HandleRequest? 还是HandleUpstreamResponse?

Handler的设置导致代码分散，维护性差。

ReadRequestHandler和ErrorWriteHandler在CodeBase中如何组织?

03. Lego架构 – Future/Promise

Future/Promise提供了基础异步机制

Continuation Passing Style
将后续逻辑作为Then的参数传递(Continuation)

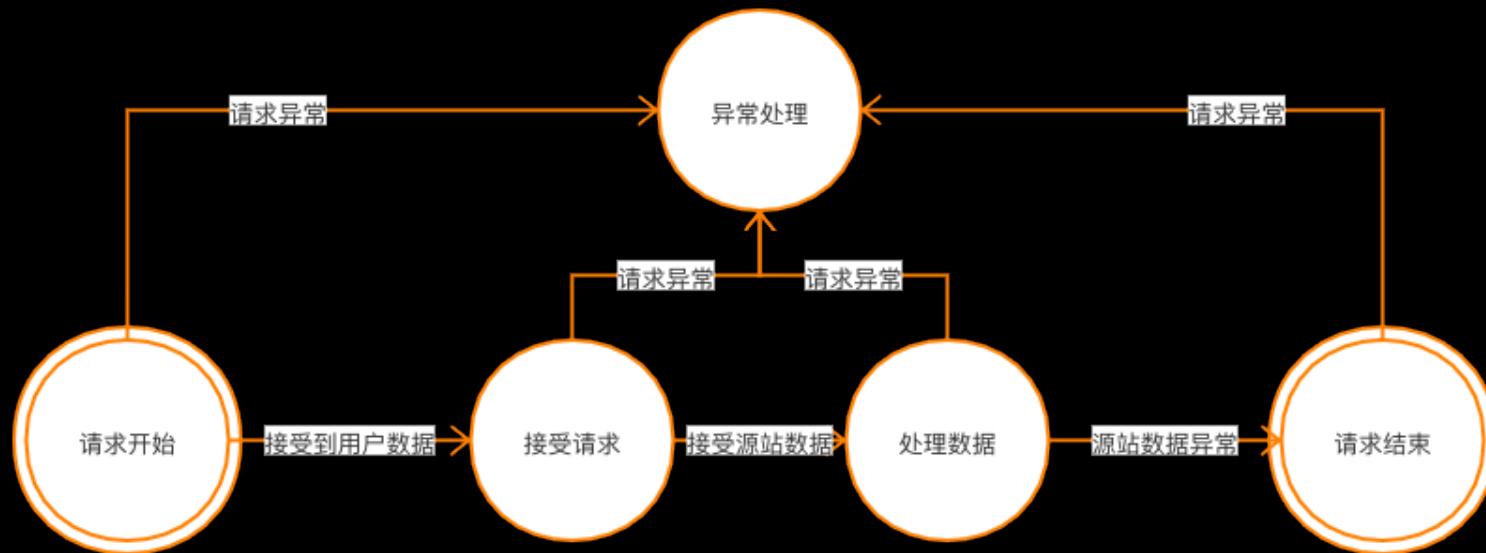
```
Future<ReturnCode> HandleRequest(Request req){  
    return req.ReadBody().Then([req](Buffer &&buf){  
        return req.WriteResponse(buf);  
    }).Finally([req](){  
        req.CleanUp();  
    })  
    return MakeReadyFuture<ReturnCode>(OK);  
}
```

通过返回Future，后续回调函数使用Then挂载，将整个应用逻辑串联起来

任何情况下，Finally都会被调用处理未捕捉异常和资源清理

03. Lego架构 – Future/Promise

状态机



CPS



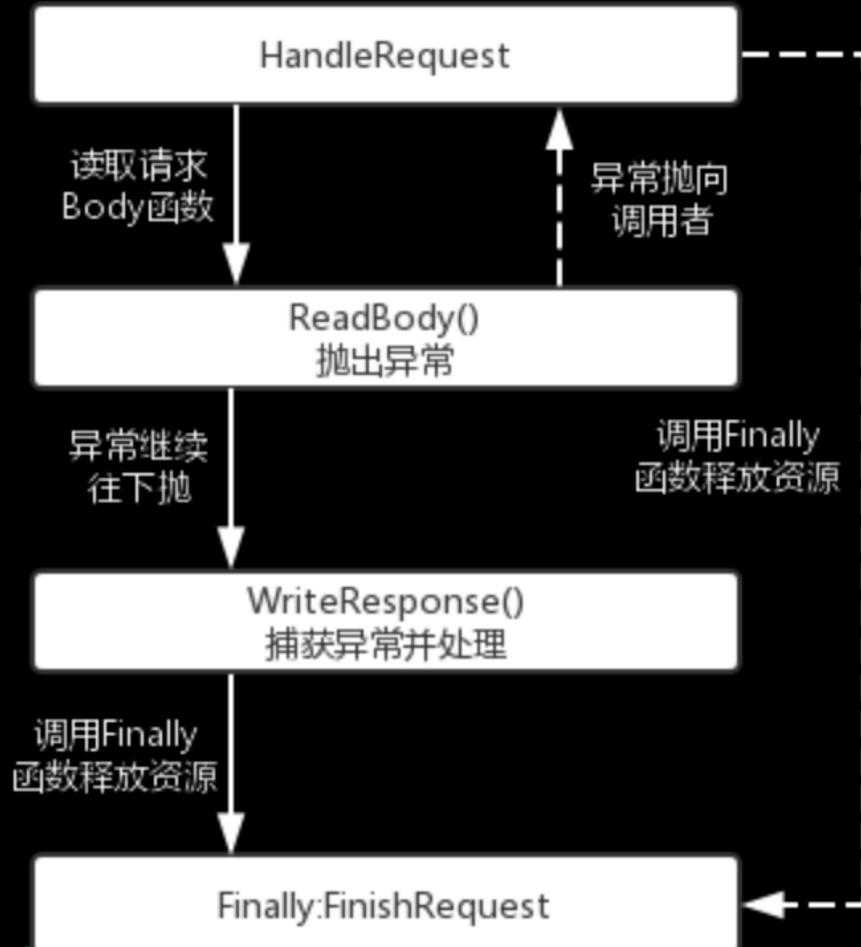
03. Lego架构 — 异常处理

自定义Exception结构：降低Exception处理开销。

1. C++目前的Exception处理涉及部分锁，以及复杂的Unwind, 查表等过程。性能开销比较大；
2. 自定义轻量化Exception架构，仅包含处理异常必要信息。

异常处理流程：NWS的exception支持finally语义和轻量级catch语义。

1. 顺着Then链路向下抛，而非向上抛；
2. 大部分业务流程并不关心后续发生的异常，反而是后续流程更关心之前发生异常；
3. Finally负责处理未捕捉异常，清理资源。



03. Lego架构 — 蜕变

```
void HandleRequest(Request req)
{
    //do something...

    req.SetReadHandler(ReadRequestHandler);
    req.SetWriteHandler(ErrorWriteHandler);
}

void HandleUpstreamResponse(Request req)
{
    //do something...
    req.SetReadHandler(ReadUpstreamResponseHandler);
    req.SetWriteHandler(ErrorWriteHandler);
}

int ErrorWriteHandler(Request req)
{
    // Something went wrong. How did I get here?
}
```

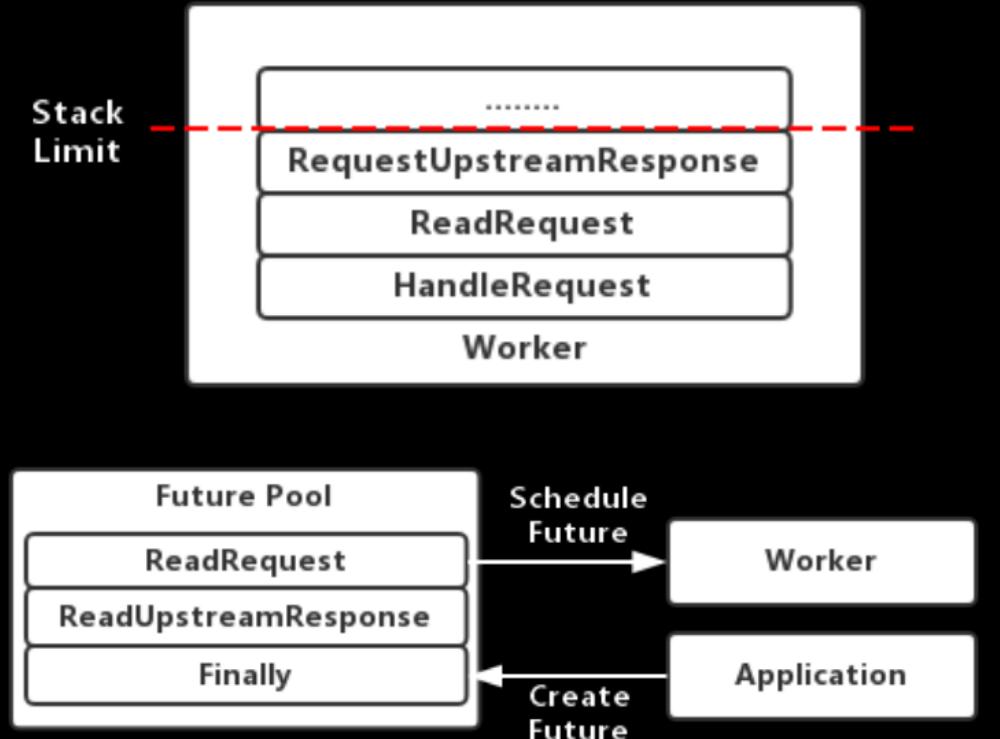


```
void HandleRequest(Request req)
{
    //do something...
    return req.ReadRequest.Then([req](){
        // do something...
        return req.ReadUpstreamResponse();
    }).Finally([
        // clean up...
    ]).GetValue();
}
```

- 接近Single Thread的编程模式，代码有更强的可读性和可维护性。
- 不需要维护额外的栈信息，没有任何额外的性能开销。
- 相对于异步调用，模块的扩展性更加灵活

03. Lego架构 – Revisited Continuation Again

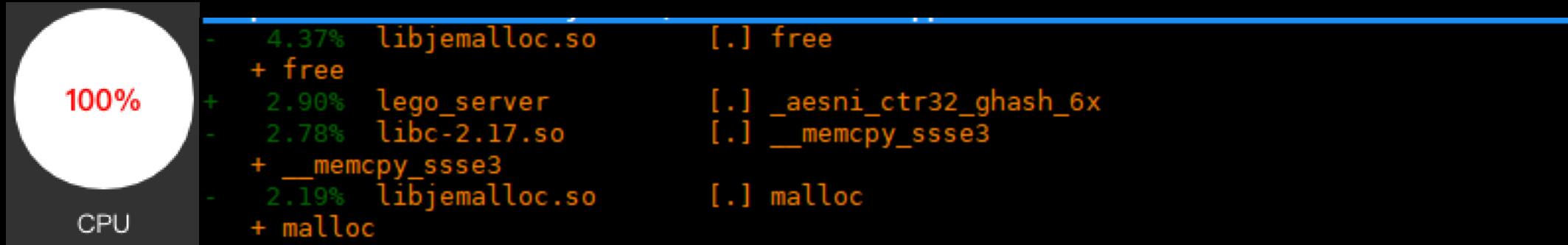
```
void HandleRequest(Request req)
{
    //do something...
    return req.ReadRequest().Then([req](){
        // do something...
        return req.ReadUpstreamResponse();
    }).Finally[]{
        // clean up...
    }).GetValue();
}
```



- Scheduler负责对Future进行优先级调度
- Future Folding, 否则由于Future的串联导致栈空间不足

03. Lego架构 – 内存管理

- 大量的Future/Promise对象创建导致内存开销增加



程序执行过程中会创建大量的Future/Promise结构，导致内存占用持续上涨

解决方案：

- 替换malloc库，如jemalloc, tmalloc等
- 每个进程独立TLS内存池，自行管理内存分配

- 局部变量holder问题，编程易出错

03. Lego架构 – 内存管理

- 大量的Future/Promise对象创建导致内存开销增加
- 局部变量holder问题，编程易出错

```
void DoWithoutHolder(Request req){  
    User test = ParseUser(req);  
  
    req.ReadRequest().Then([req, test](){  
        return req.ReadUpstreamResponse(test); // test可能已经被释放了  
    });  
}
```

```
void DoWithHolder(){  
    std::shared_ptr<User> test = std::make_shared(std::move(ParseUser(req)));  
  
    req.ReadRequest().Then([test, req](){  
        return req.ReadUpstreamResponse(test);  
    });  
}
```

03. Lego架构 – 内存管理

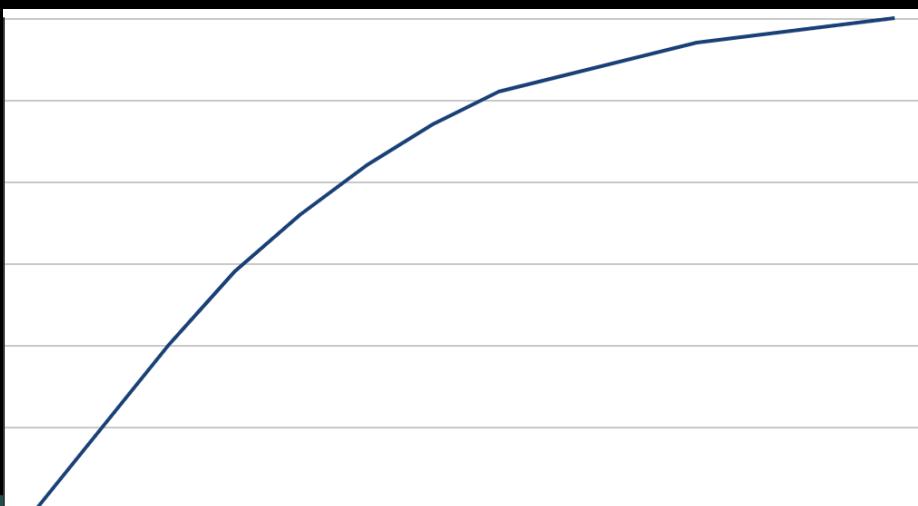
- 大量的Future/Promise对象创建导致内存开销增加
- 局部变量holder问题，编程易出错

```
void DoWithoutHolder(Request req){  
    User test = ParseUser(req);  
  
    req.ReadRequest().Then([req, test](){  
        return req.ReadUpstreamResponse(test); // test可能已经被释放了  
    });  
}
```

```
void DoWithHolder(){  
    std::shared_ptr<User> test = std::make_shared(std::move(ParseUser(req)));  
  
    req.ReadRequest().Then([test, req](){  
        return req.ReadUpstreamResponse(test);  
    });  
}
```

03. Lego架构 – Shared-Nothing

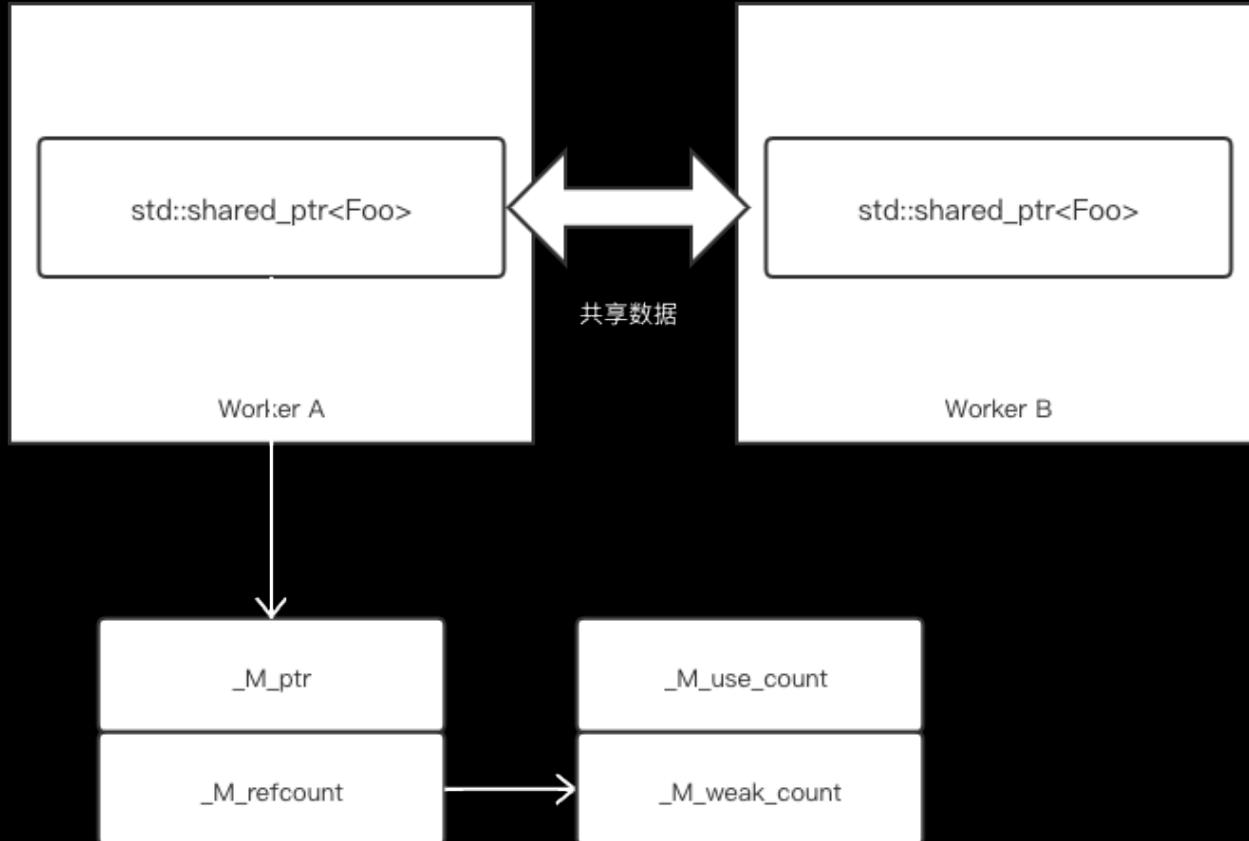
- CPU核数变为原有4倍，服务器性能却只增加2倍
- 单机服务器性能无法线性扩展，cache和锁等问题导致核数增加并不会带来期望的性能提升
- 每个核心独立运行完整程序才能达到性能平行扩展



Numbers Everyone Should Know (Jeff Dean, Google)²

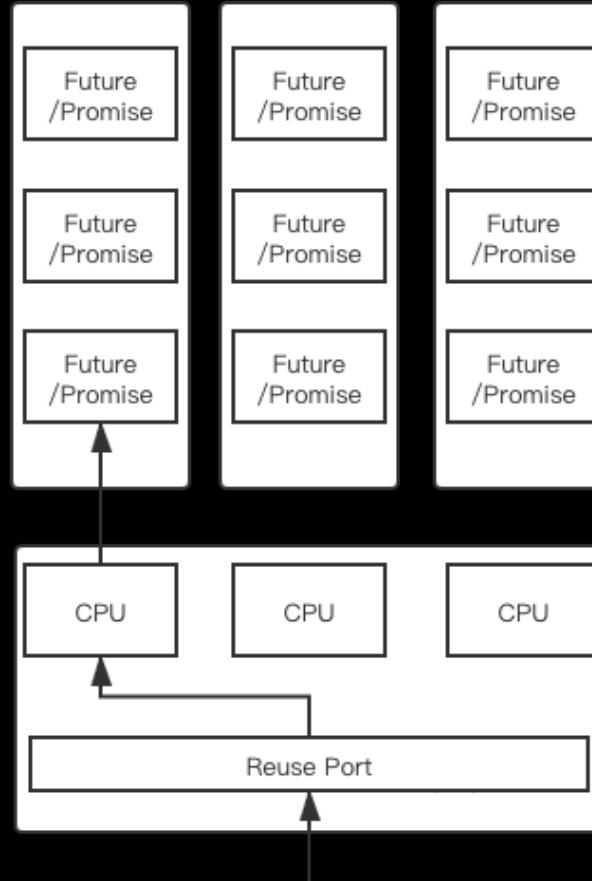
| | |
|--------------------------------------|-------------|
| • L1 cache reference: | 0.5 ns |
| • Branch mis-predict: | 5 ns |
| • L2 cache reference: | 7 ns |
| • Mutex lock/unlock: | 25 ns |
| • Main memory reference | 100 ns |
| • Compress 1K Bytes with Zippy | 3000 ns |
| • Send 2K Bytes over 1 GBPS network | 20000 ns |
| • Read 1 MB sequentially from memory | 250000 ns |
| • Round trip within data center | 500000 ns |
| • Disk seek | 1000000 ns |
| • Read 1MB sequentially from disk | 2000000 ns |
| • Send one packet from CA to Europe | 15000000 ns |

03. Lego架构 – Share Pointer



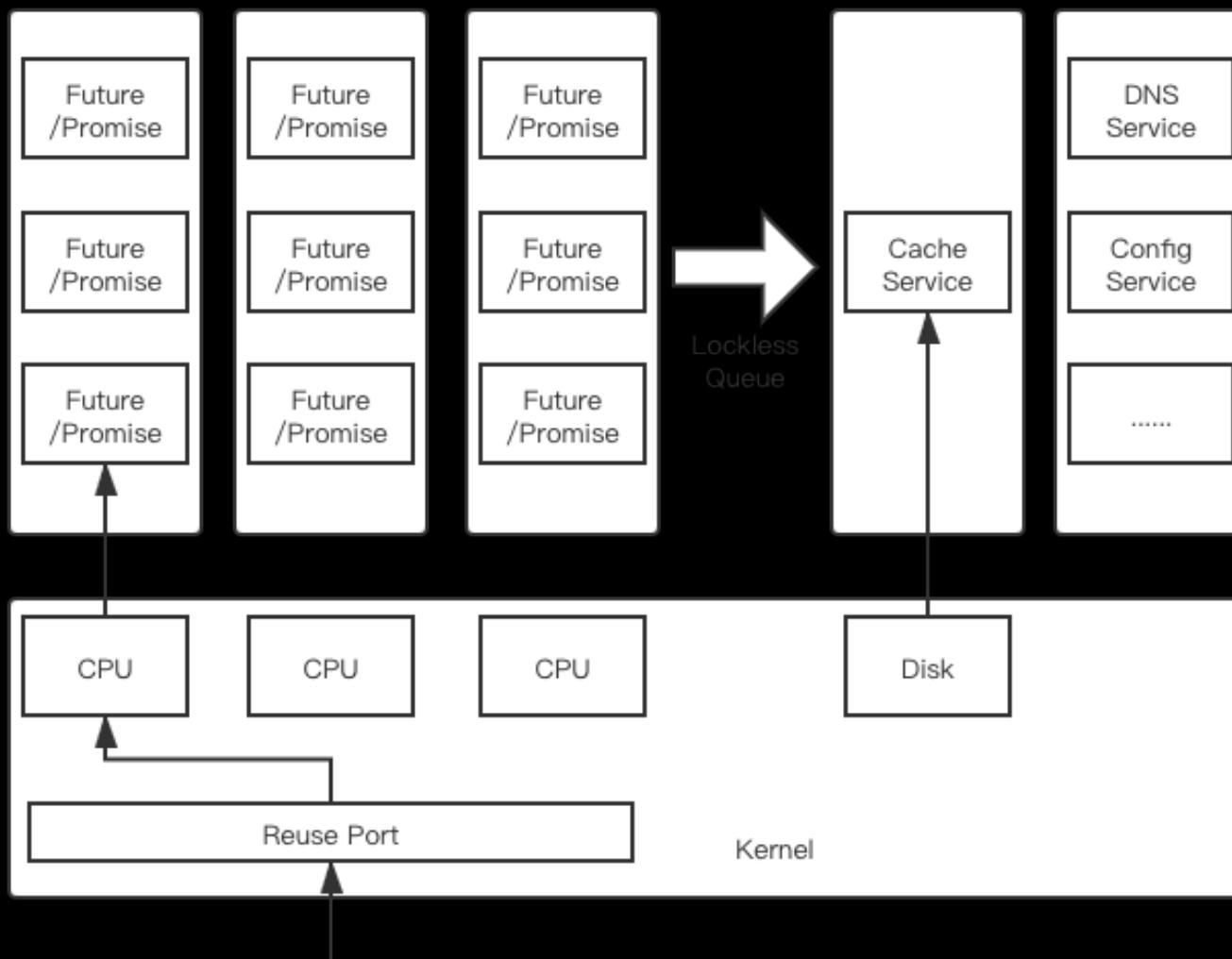
- std::shared_ptr并不智能
- Atomic reference counter带来的性能开销不可控，部分场景下导致CPU高负载
- 跨线程/进程共享数据时，仅有引用计数是 thread-safe，指针操作并不安全
- 不同场景下需要不一样的智能指针
- 单进程中引用计数使用替代atomic变量
- 类似atomic_load解决方案，内部使用mutex数组来解决并发问题

03. Lego架构 – Shared-Nothing



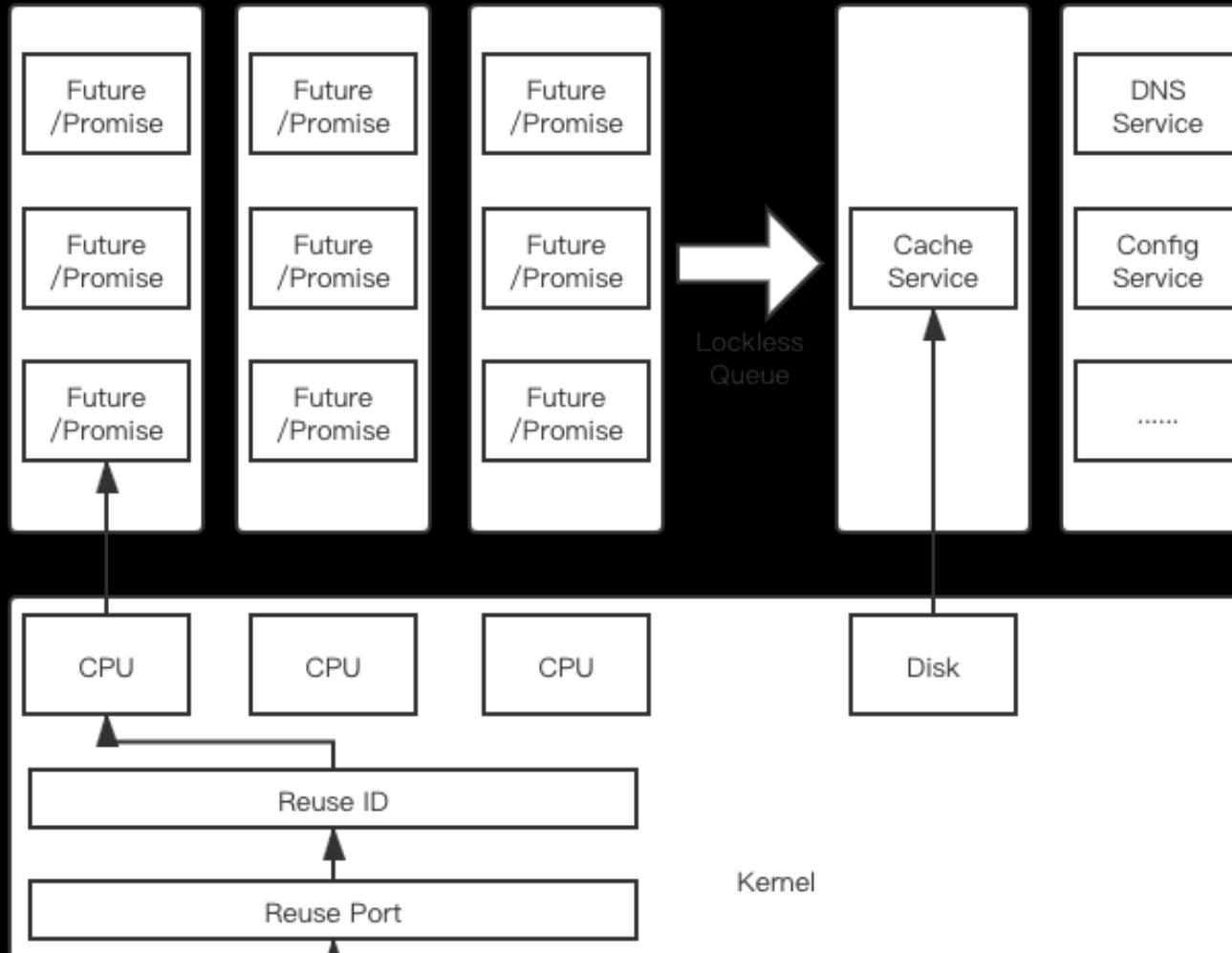
- 每个进程/线程互相之间尽可能避免共享全局数据规避锁和在不同CPU间读写数据时的Cacheline影响
- 通过Linux Kernel Reuse Port特性，自动将链接分发至不同的进程，之后对应链接的请求完全由对应进程/线程处理
- 每个进程绑定固定的CPU，规避进程在不同CPU间迁移
- 服务器性能可以达到近乎平行扩展的能力
- 总是有需要全局共享的数据/处理复杂度较高的任务，如何处理？

03. Lego架构 – Shared-Nothing



- 每个进程/线程互相之间尽可能避免共享全局数据规避锁和在不同CPU间读写数据时的Cacheline影响
- 通过Linux Kernel Reuse Port特性，自动将链接分发至不同的进程，之后对应链接的请求完全由对应进程/线程处理
- 每个进程绑定固定的CPU，规避进程在不同CPU间迁移
- 服务器性能可以达到近乎平行扩展的能力
- 总是有需要全局共享的数据/处理复杂度较高的任务，如何处理？

03. Lego架构 – Shared-Nothing



- Quic类型的请求无法通过Reuse Port进行正确的路由
- 客户端IP和Port变更后，仍然能够转发到对应的线程，解决用户使用场景变更问题，比如4G切换到Wifi
- Quic中使用了一个Connection ID作为标识
- 根据对应ID进行请求路由

03. Lego架构 — 总结

不断攀登

高性能

- Shared-Nothing
- Shared_Ptr
- Exception Handle
- 内存管理

选好工具

可维护

- Future/Promise
- Continuation Passing Style
- 代码规范
- 模块隔离

04. 未来展望 — 最完美代码

```
void HandleRequest(Request req)
{
    //do something...
    return req.ReadRequest.Then([req] () {
        // do something...
        return req.SelectUpstream().Then([req] () {
            // do something...
            return req.CreateUpstream().Then([req] () {
                // do something...
                return req.ReadUpstreamResponse();
            });
        });
    }).Finally([]{
        // clean up...
    }).GetValue();
}

auto HandleRequest(Request req)
{
    // do something...
    req.ReadRequest();
    // do something...
    req.SelectUpstream();
    // do something...
    req.CreateUpstream();
    // do something...
    auto result = req.ReadUpstreamResponse();
    // clean up...
    return result;
}
```

- 完全Single Thread的写法，避免了Callback Hell。
- 编译器有更多的优化空间，进一步的提升程序性能。

04. 未来展望 – C++20与服务器

- C++20 Coroutine

```
task<> tcp_echo_server() {
    char data[1024];
    for (;;) {
        size_t n = co_await socket.async_read_some(buffer(data));
        co_await async_write(socket, buffer(data, n));
    }
}
```

`operator co_await(static_cast<Awaitable&&>(awaitable))` for the non-member overload)

```
auto switch_to_new_thread(std::jthread& out) {
    struct awaitable {
        std::jthread* p_out;
        bool await_ready() { return false; }
        void await_suspend(std::coroutine_handle<> h) {
            std::jthread& out = *p_out;
            if (out.joinable())
                throw std::runtime_error("Output jthread parameter not empty");
            out = std::jthread([h] { h.resume(); });
            // Potential undefined behavior: accessing potentially destroyed *this
            // std::cout << "New thread ID: " << p_out->get_id() << '\n';
            std::cout << "New thread ID: " << out.get_id() << '\n'; // this is OK
        }
        void await_resume() {}
    };
    return awaitable{&out};
}
```

- 可以最大化利用C++编译器优化能力
- 自动管理Coroutine的上下文
- 目前要求整链路都是Awaitable
 - func1->func2->func3
- 实际性能和内存开销待验证

04. 未来展望 – 待优化点

- 云原生
 - 服务器开包即用
- Kernel Bypassing
 - DPDK/XDP
 - SPDK/io_uring
- 软硬结合
 - 解密卡
 - FPGA
- 可中断服务支持
 - Preemptive task

The End

张超

软件技术咨询师



软件技术咨询师，有近十年的大规模嵌入式实时系统 C/C++ 开发重构经验，帮助多家公司实施大规模嵌入式重构、性能优化，微服务架构转型设计与实施，自动化测试框架的设计。专注于系统软件的领域建模设计、重构、自动化测试以及极致开发体验。

主办方：

Boolan
高端 IT 咨询与教育平台

CPP-Summit 2020

张超
软件咨询顾问

C++ Modules 与大规模物理设计

议程

1 C++ 物理设计的困境

2 C++ Modules介绍

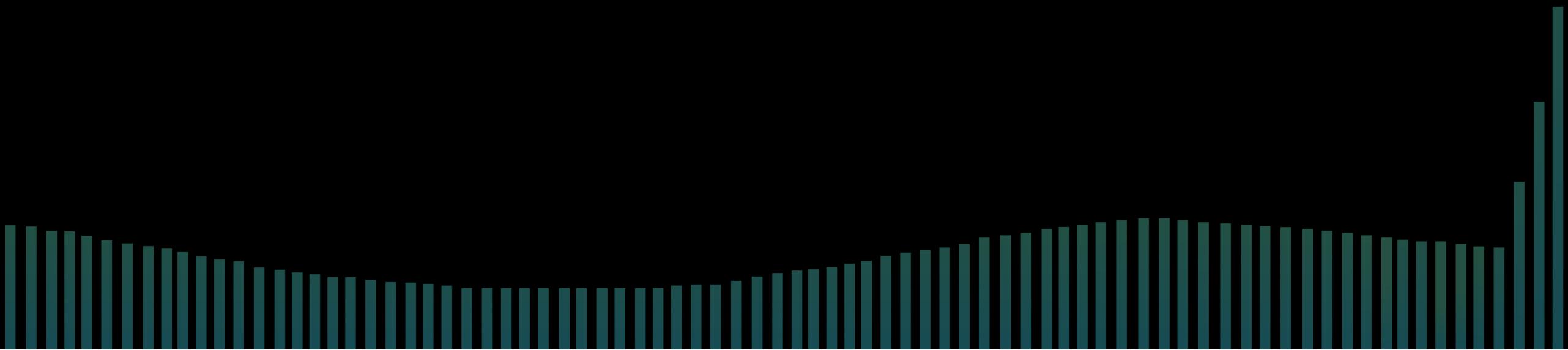
3 Modules 改善的物理设计

4 Modules 的现状与发展

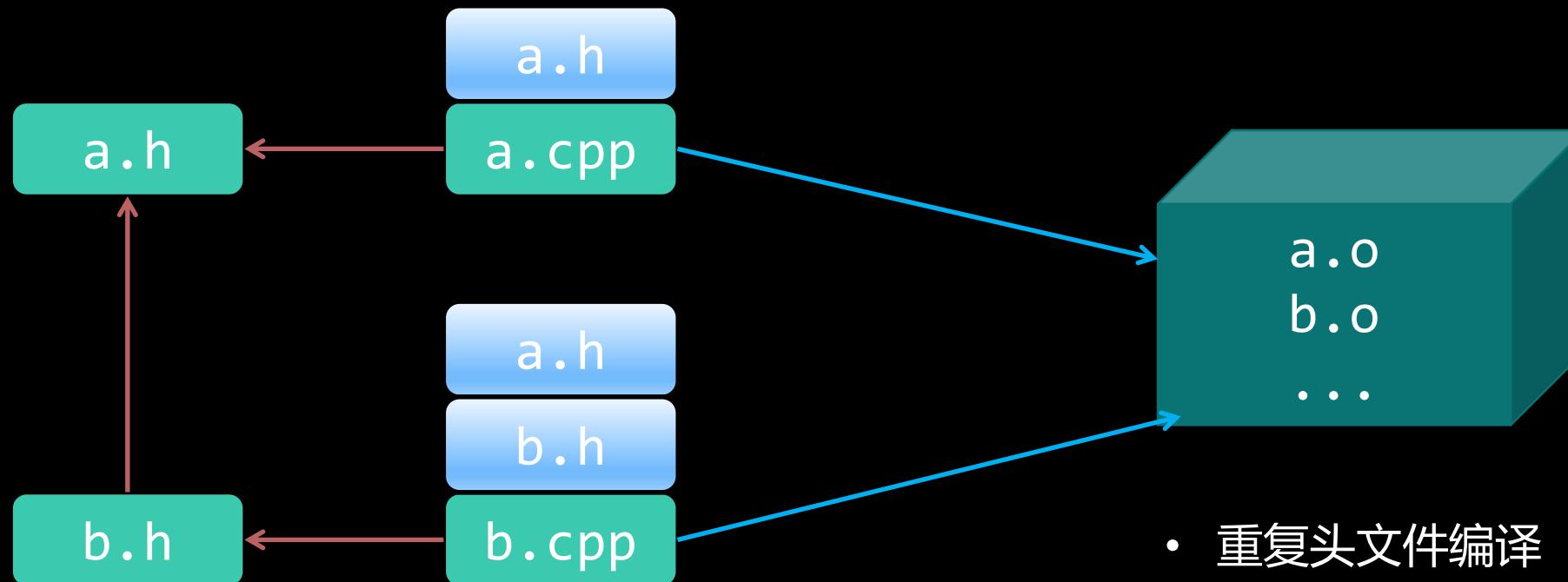
5 遗留系统的改造建议

01

C++物理设计的困境

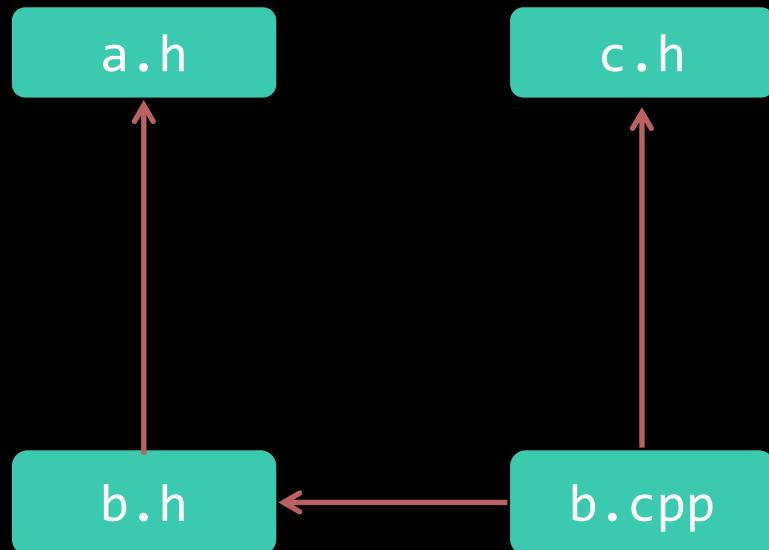


编译时间长



- 重复头文件编译
- 链接无序查找

可能对程序造成错误和干扰



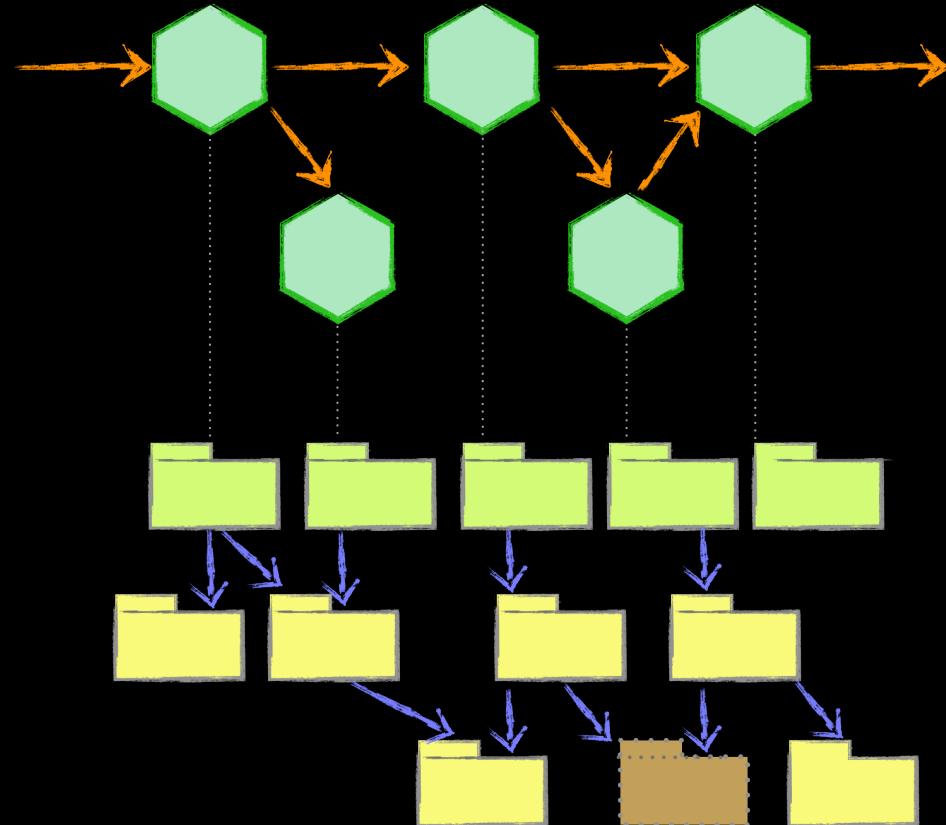
- 头文件不自满足
- 宏定义经由头文件传播导致的错误替换
- `#include` 的顺序导致解释差异
- 符号的重名

难以理清的依赖关系



- A `include` B 不知道具体用了哪些东西，
- A `include` B 不知道间接依赖其他的头文件
- A `link` B 只在构建脚本中体现
- `Extern` 的使用让依赖关系更无处可寻

物理设计的目标

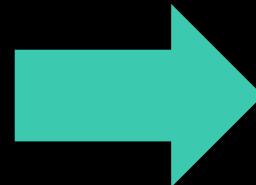


- 编译：可以高效正确的执行
- 开发：更好的服务于开发效率，不容易出错
- 架构设计：直观的反映逻辑设计和依赖关系

物理设计困难根因分析

现有手段：

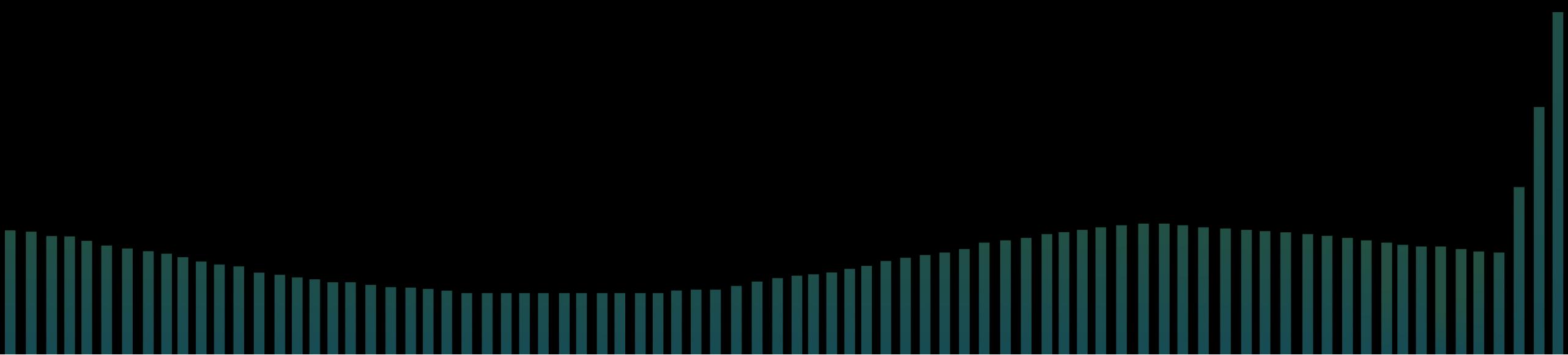
- 编译时间长：物理设计原则，PCH
- 宏定义破坏：无能为力
- 符号冲突：static/namespace
- 依赖管理：物理设计原则



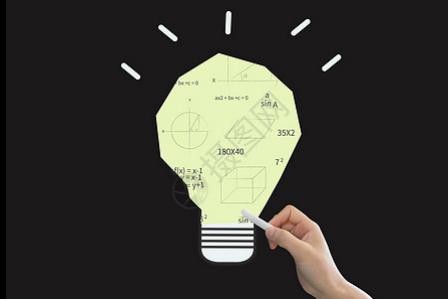
- 预处理机制存在缺陷
- 缺乏可见性控制手段

02

C++ Modules介绍



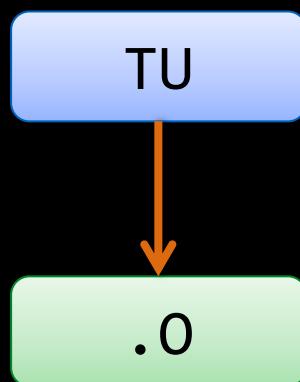
MODULES 的解题思路



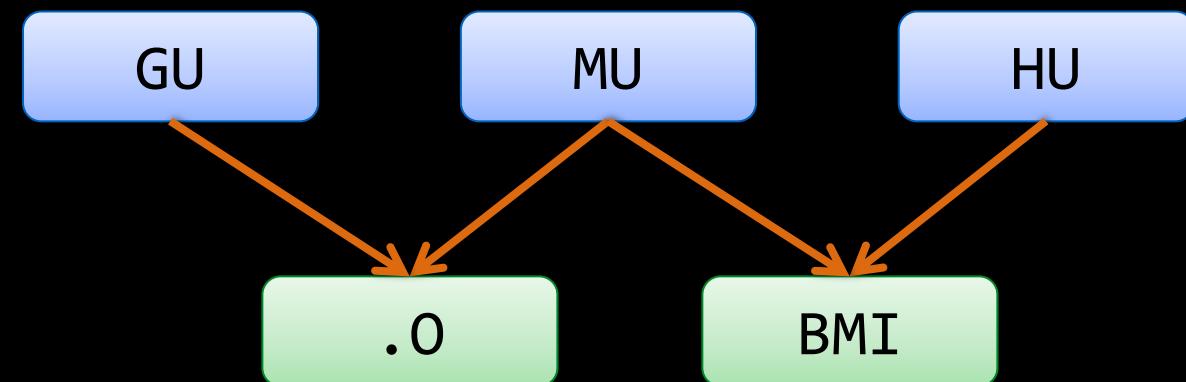
- 新的引用方式`import`与新的复用单元`module`代替`#include`与头文件
- 提供显示的控制符号的能力`export`

MODULES 新增概念

- Translation Unit



- Module Unit: `cppm/ixx/mxx/`
- Global Module : `cpp/cxx`
- Header Unit: `h.hpp` (`importable`)
- BMI (Binary Module Interface) /CMI



关键字-MODULE

```
// TU 1
export module A;
export int baz();
// TU 2
export module Foo;
export int foo() {
    return 0;
}

// TU 3
module;
#include <math.h>

module A;
int bar();
int baz() {
    return abs(bar()) + 1;
}

module :private
int bar() {
    return -1;
}
```

- 声明Module
 - 全局片段
 - 实现Module (可选)
 - 私有片段
-
- Module的范围从声明 (定义) 开始
 - Module的合法名字使用 “.” 连接
 - 全局片段中的内容，全局链接
 - 私有片段里的内容，不跨MU链接

关键字-EXPORT

```
module A;
export int bar() {
    return -1;
}

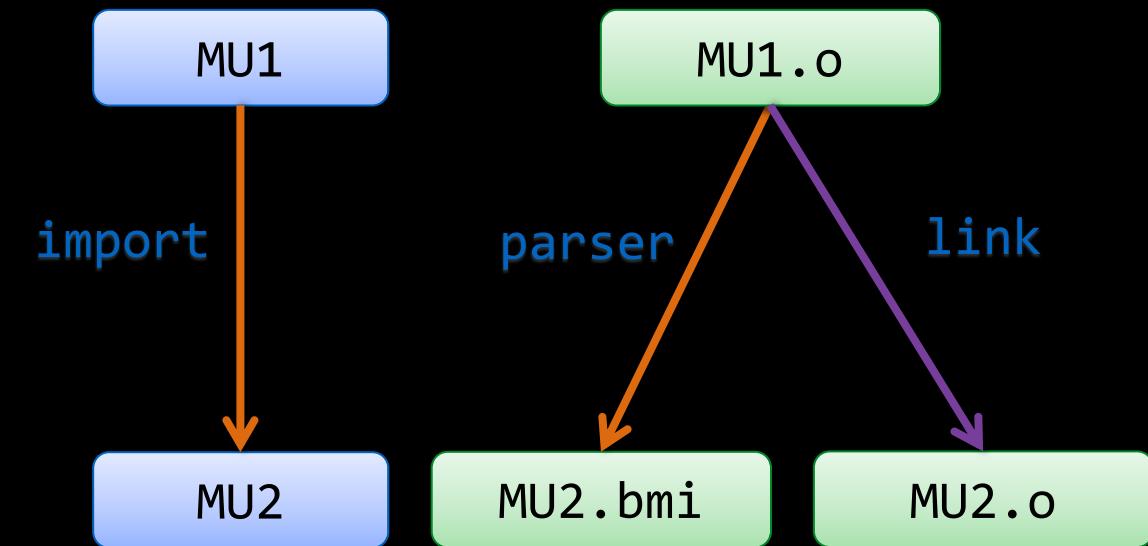
export {
    class e1 {
        int x;
        int y;
    };
    int e1;
}

namespace baz {
    export void quux();
}

export namespace bar{
    void f1();
    int v1 = 0;
}
```

- 必须有名字的才export（编译过后有名字的）；static函数，匿名namespace，不能导出
- export namespace 导出 本段 export namespace 包含的内容
- export namespace 中的部分，间接导出命名空间名字
- 支持 export block
- Module Implementation 不能包含export

关键字-IMPORT



import A::B 不是A中的B内容，A::B 一定也是个MU

import 也有循环依赖的风险。

import 不等于#include

import 2个不同module中相同的符号，仍然会冲突

import 只能在全局范围，不能在namespace里

一个MODULE UNIT的完整例子

```
//Global Fragment
modules;
#include "some.h"

//ModuleName
export module M;
export module :Internal

//Imports
import X;
import Y;

//Exports Internal Depends
struct FooBase {
};

//Exports
export struct Foo : FooBase{
};

// Private Fragment
modules:private;

//Internal Implementation
int xx(){
}
```

- 建议只放include

- 声明Module
- Import
- Export内容的依赖声明
- Export

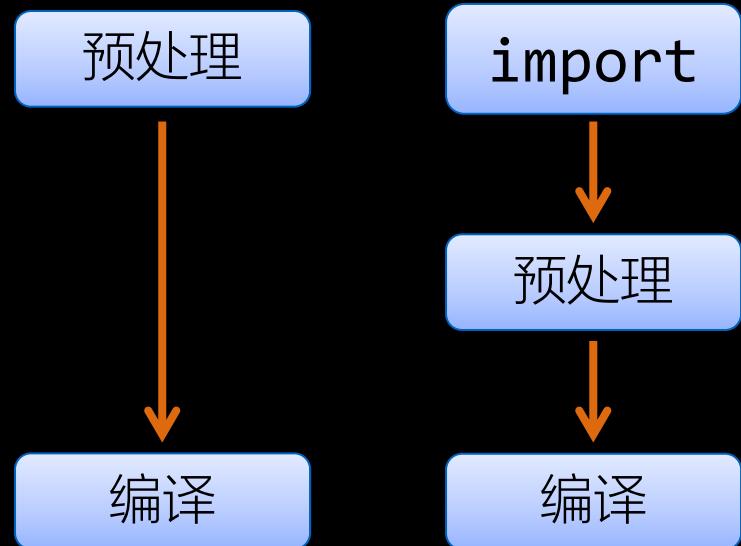
- 内部实现

MODULE UNITS 分类

```
// TU 1
export module A;
export import :Foo;
export int baz();
// TU 2
export module A:Foo;
import :Internals;
export int foo() {return 0;}
// TU 3
module A:Internals;
int bar(){ return 1;}
// TU 4
module A;
import :Internals;
int baz() { return bar() + 1;}
```

- Module Interface
- Module Implementation
- Module Partition
- 方便演进
- 进一步降低依赖
- 并行开发

MODULE 与 MACRO



- 宏的作用范围不会超过module。
- 需要共享的宏定义只能在Header Unit中，不可传递
- 不要用宏定义modules的关键字

MODULE 重定向

```
// TU 1
export module FooWrapper;
export import Foo;
export import Foo1;
export import Foo2;
export import Foo3;
```



聚合

常用作库的接口，方便并行开发后整合。

```
// TU2
export module Foo;
export import Foo:doo;
```

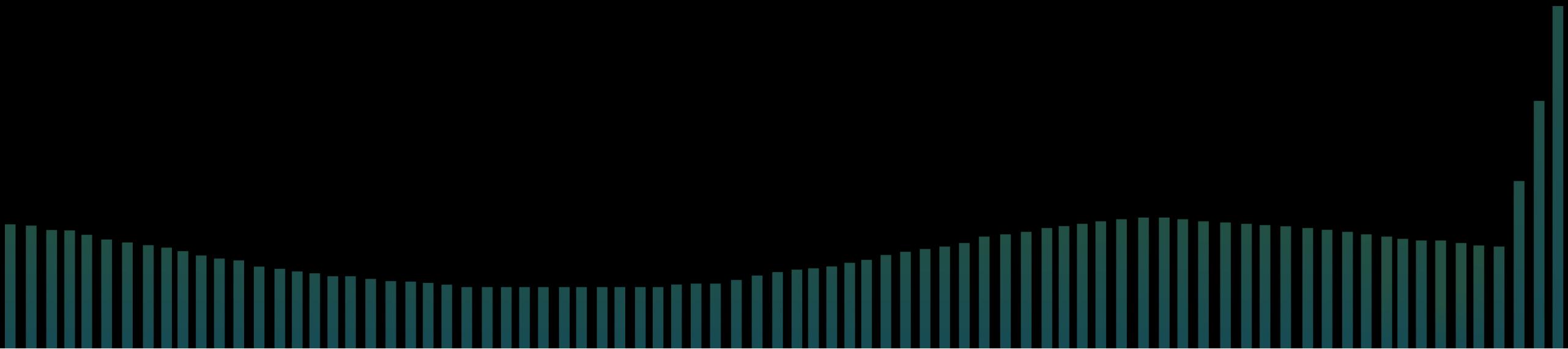


封装

为了特定用户封装部分接口；
非接口module最好不要
export import

03

Modules改善的物理设计



MODULES 带来编译速度提升

The screenshot shows two side-by-side code editors. The left editor displays a file named 'm1.ixx' containing module declarations and definitions. The right editor displays a file named 'm1.cpp' containing standard C++ code. Both editors show similar output panes at the bottom.

```

Left Editor (m1.ixx):
1 module;
2 #include <iostream>
3 export module math;
4
5 #define MAX_NUM 102

```

```

Right Editor (m1.cpp):
1 #include <iostream>
2
3 #define MAX_NUM 102
4
5 struct Pair {
6     int x;
7     int v;
8 }

```

Output panes (Bottom):

```

Left Editor Output:
1> 5 毫秒 ComputeCustomBuild0
1> 6 毫秒 InitializeBuildStat
1> 6 毫秒 GetCopyToOutputDir
1> 7 毫秒 AfterResourceCompile
1> 7 毫秒 CoreClean
1> 7 毫秒 MakeDirsForLink
1> 9 毫秒 ComputeCLOutputs
1> 18 毫秒 FixupCLCompileOptions
1> 340 毫秒 CoreCppClean
1> 974 毫秒 Link
1> 15171 毫秒 SetModuleDependencies
1> 21252 毫秒 ClCompile

```

```

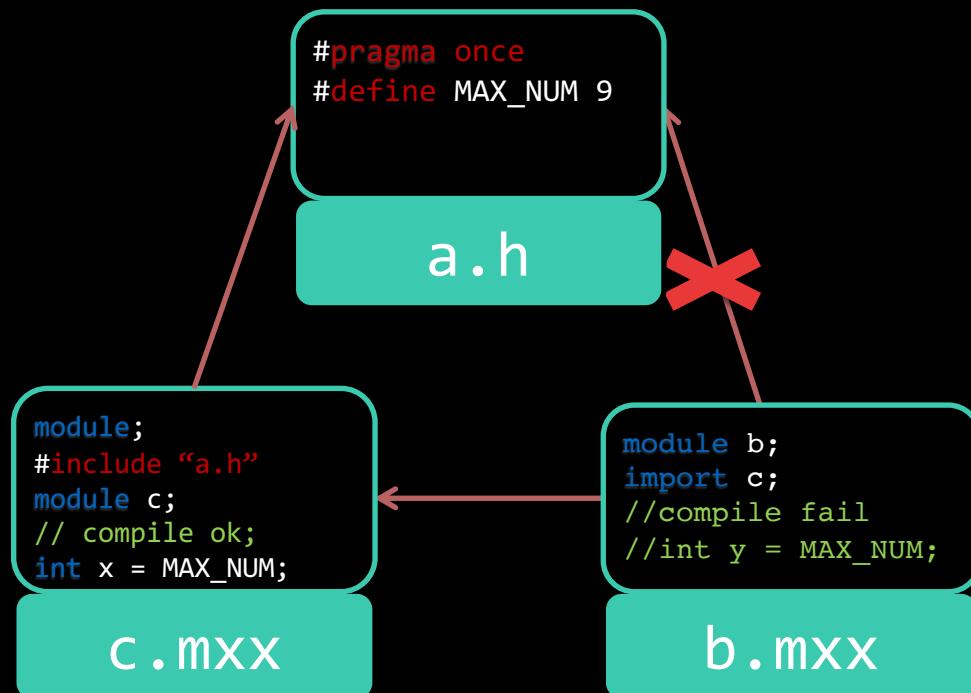
Right Editor Output:
1> 14 毫秒 _PrepareForClean
1> 15 毫秒 SetBuildDefaultEnvironmentVariables
1> 15 毫秒 SatelliteDl1sProjectOutputGroup
1> 18 毫秒 ResolveProjectReferences
1> 23 毫秒 WarnCompileDuplicatedFilename
1> 29 毫秒 _CheckForInvalidConfigurationAndPlatform
1> 34 毫秒 CoreClean
1> 51 毫秒 FindReferenceAssembliesForReferences
1> 90 毫秒 CreateRecipeFile
1> 116 毫秒 CoreCppClean
1> 891 毫秒 Link
1> 89977 毫秒 ClCompile

```

- 减少了头文件的重复编译；
- 确定方向的链接；
- 对于头文件很小，不包含标准头文件的系统，或已经PCH/UNIT SOURCE优化的，提升有限。

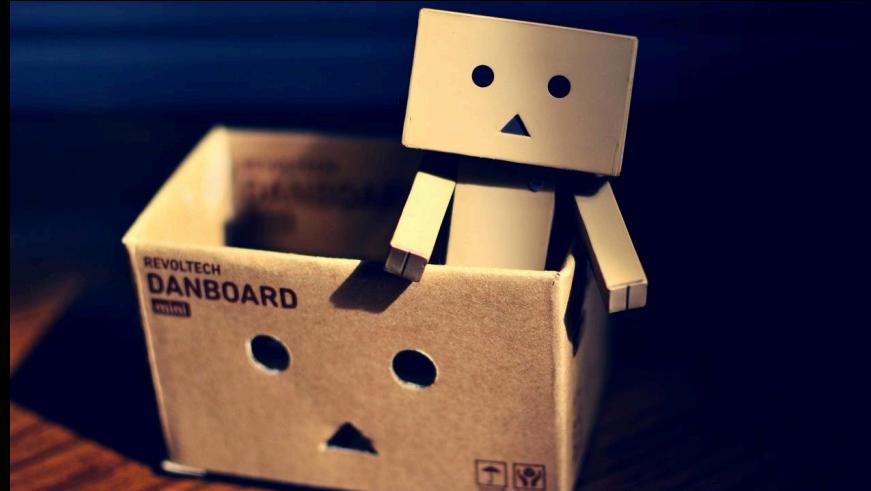
"100个cpp 包含 (include) 1个头文件 " vs "100个ixx 引入 (import) 1个ixx"

MODULES 为依赖管理带来的改善



- 宏定义不能跨Module传递
- 可以轻松从源码获取的准确的依赖关系
- 选择性可见，选择性可达
- 符号链接有了确定的指向

MODULES 带给包管理的机遇和挑战



- 大大提高了接口包的稳定性
- 源码与二进制之外，新的复用单元（BMI）
- 代码内置的依赖关系，需要与描述文件中的内容一致，最好避免重复声明
- Modules的命名，目录应该与包管理路径的结合。

MODULES没法代替你做出良好的物理设计

物理依赖与逻辑依赖保持一致

一致性

单一职责

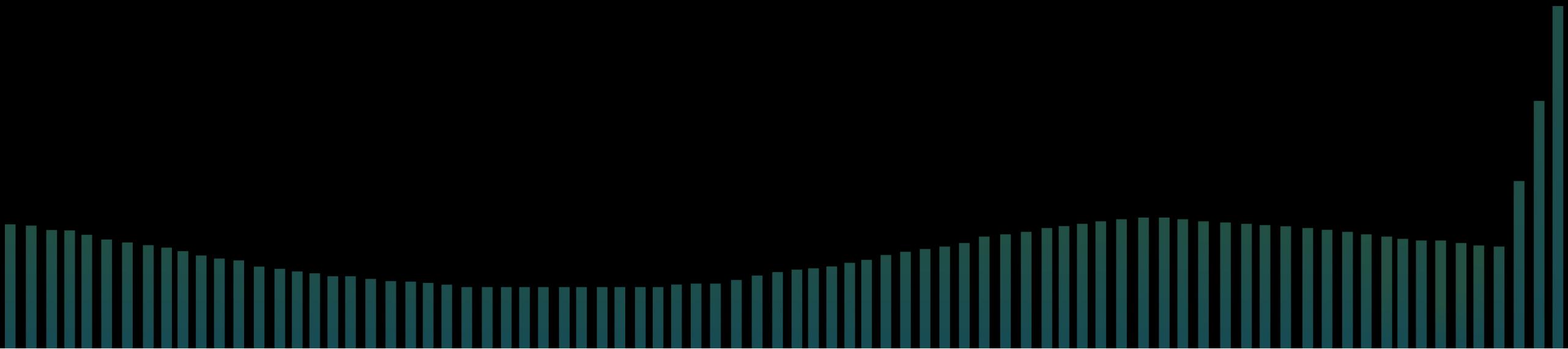
最小可见

只有一类导致模块变化的原因

尽量少的暴露接口和结构

04

Modules的现状与发展



MODULES 编译器标准现状



- 编译器 : clang , gcc(branch) , msvc
- 构建工具 : cmake , ninja , build2...
- IDE : eclipse , clion , vs2019...

Modules对整个C++的生态链都是颠覆式的改变

MODULES 对比其他语言

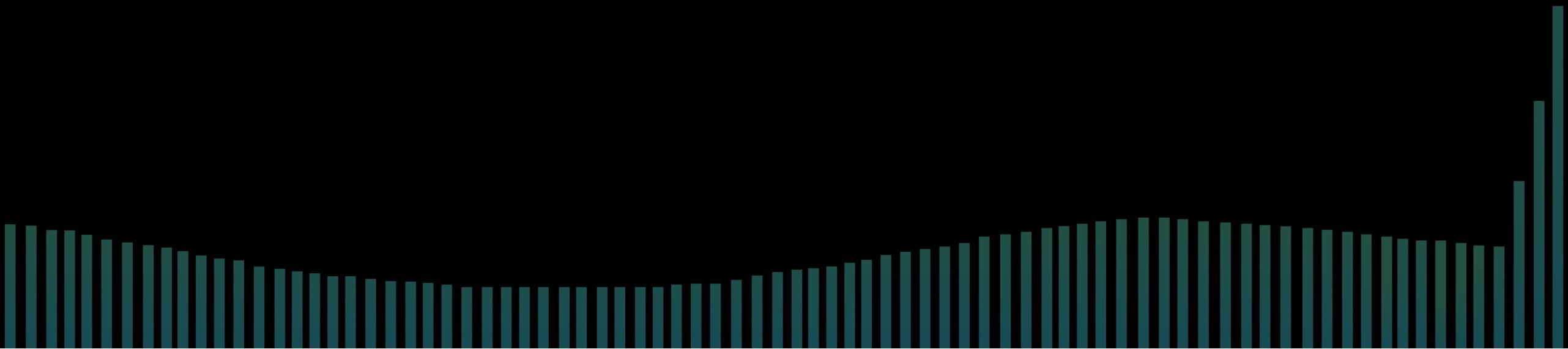
| 语言 | 模块定义 | 导出 | 导入 | 模块描述文件 |
|--------|----------------|-------------|-----------------|-------------------|
| C++20 | module | export | import | 内置 |
| python | 目录/文件 | all | import | 外置.__init__.py |
| rust | mod | pub | use | 内置 |
| golang | package | all | import | 外置.mod |
| java | module/package | exports/all | requires/import | module-info.class |

MODULES 遗留问题

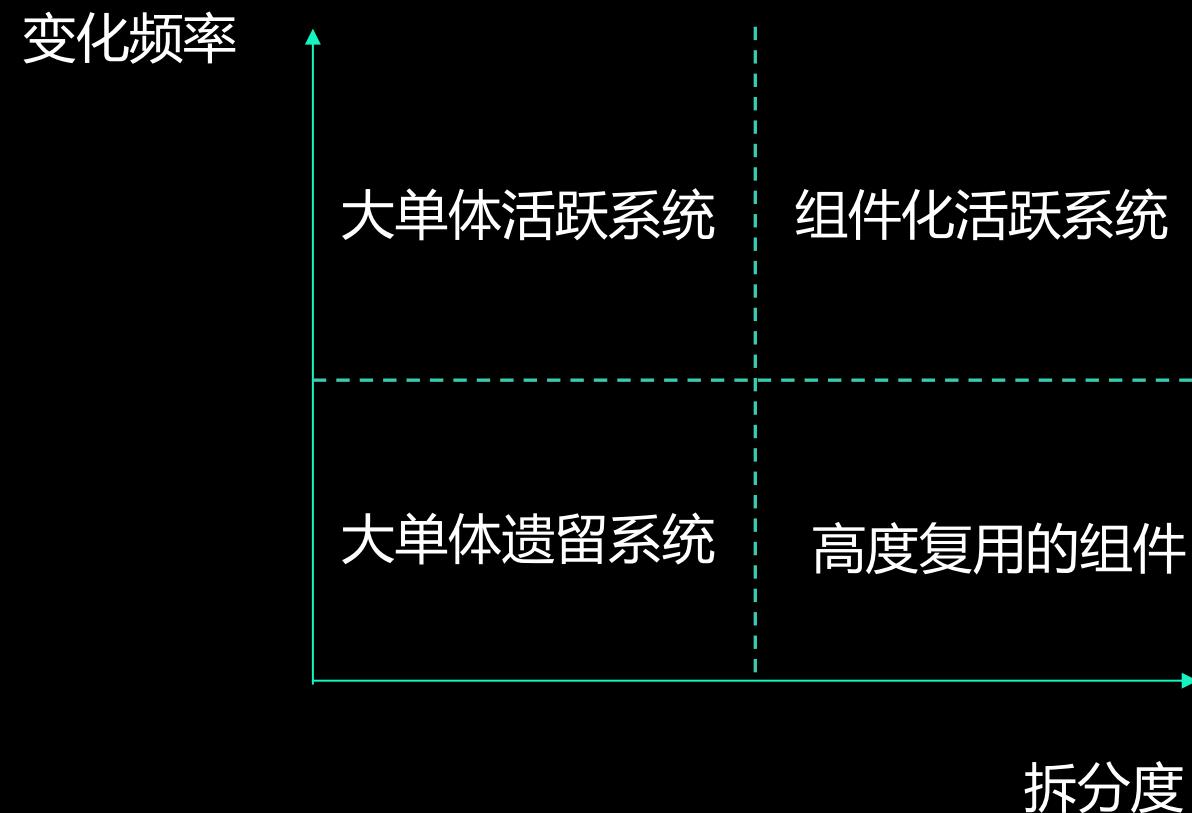
- 灵活的命名带来的困扰：
 - 可以多个“.”连接，但没有层次的语义，
 - module的名字与文件名没有任何关系。
- 不支持单个符号的import，
- 无用的import的可以优化处理
- 依赖生成和搜索性能，还有待优化
- 不同平台构建的BIM仍然无法通用

05

遗留系统的改造建议



如何演进到MODULES的一些原则



原则：

- 适可而止
- 小步安全的重构

场景1:大型单体遗留系统-建议编译提升

分析：

- 规模大重构风险高
- 需求少验证成本高

重构步骤：

1. 替换或拆分头文件为Header Unit(importable)
2. 替换依赖它的原有include为import，或增加新的import
3. 构建验证

场景2:大单体活跃系统-建议MODULE拆分

分析：

- 需求活跃，可随需求验证
- 生命周期长，对响应变化有诉求

重构步骤：

1. 找到可复用的单元，逐步提取出到新的Module文件中，`export`相应的接口。
2. 原有的代码需要用到Module的，`import` 新的Module。
3. 测试验证

场景3:组件化活跃系统-建议合理化依赖

分析：

- 组件间依赖可控，依赖双方可同时修改
- 有相对清晰的边界

重构步骤：

1. 修改include文件为Module Interface，export相应的接口，cpp改为Module Implement。
2. 调整组件对外部库的inlcude为聚合Module Interface。
3. 替换依赖该组件的为import

场景4：高度复用的组件-并存过渡接口

分析：

- 对库的使用者没法控制同步修改
- 希望部分用户享受Module

重构步骤：

1. 替换或拆分头文件为Header Unit(importable)
2. 实现文件全部为Global Unit。
3. 对外提供的接口分别以include和import形式呈现

MODULES大势所趋，理当积极准备

- 好的物理设计距离Modules之差一步之遥
- 准备好宏定义，按照Modules风格设计文件，切换编译器时搜索替换
- 保持Module文件命名和路径与文件系统保持一致
- 借助包管理工具熟悉并设计Module的层次以及梳理依赖关系
- 头文件转换工具Mapping

谢谢

张汉东

资深Rust专家，《Rust编程之道》作者



曾在互联网行业沉浮十余载，先后效力于电商、社交游戏、广告和众筹领域。作为资深咨询师，先后为华为、思科、平安科技、闪迪等公司提供咨询服务。于2015年开始研究Rust语言，并参与国内Rust社区的管理和运营，国内最早一批Rust研究者。在2019年初出版技术畅销书《Rust编程之道》，深受好评。

主办方：

Boolan
高端IT咨询与教育平台

CPP-Summit 2020

张汉东

《Rust编程之道》作者
企业独立咨询顾问

Rust系统级开发 的优势与劣势

议程

1 自我介绍

一个简单的自我介绍

2 时代视角下的系统级开发

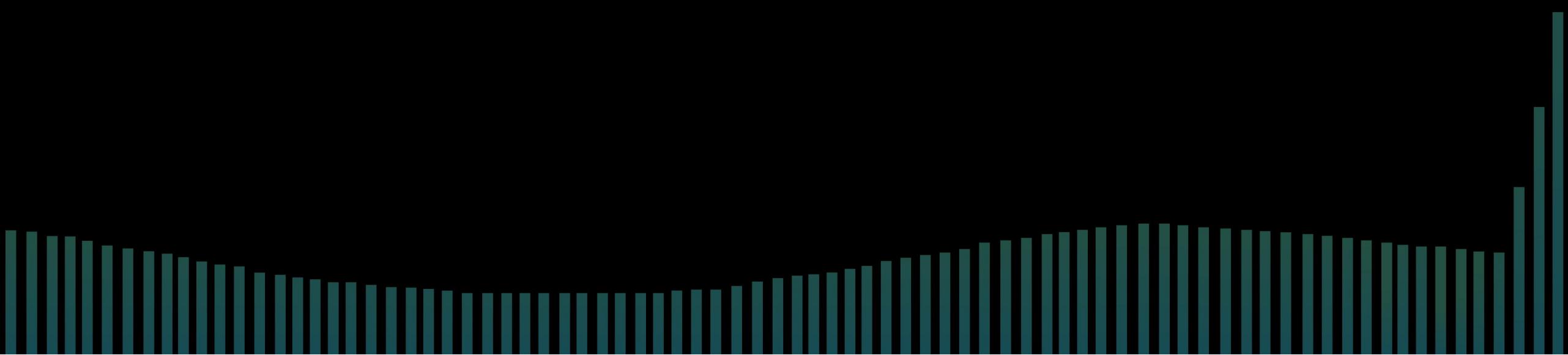
从不同时代的角度去看待系统级开发的变迁

3 Rust 语言的兴起

介绍 Rust 语言

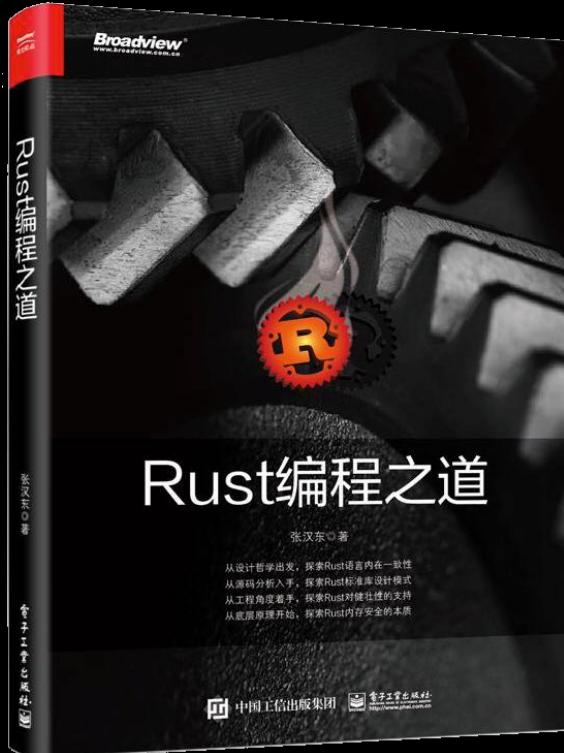
4 Rust 语言的优势与劣势

高屋建瓴式分析 Rust 语言的优势与劣势



01

自我介绍



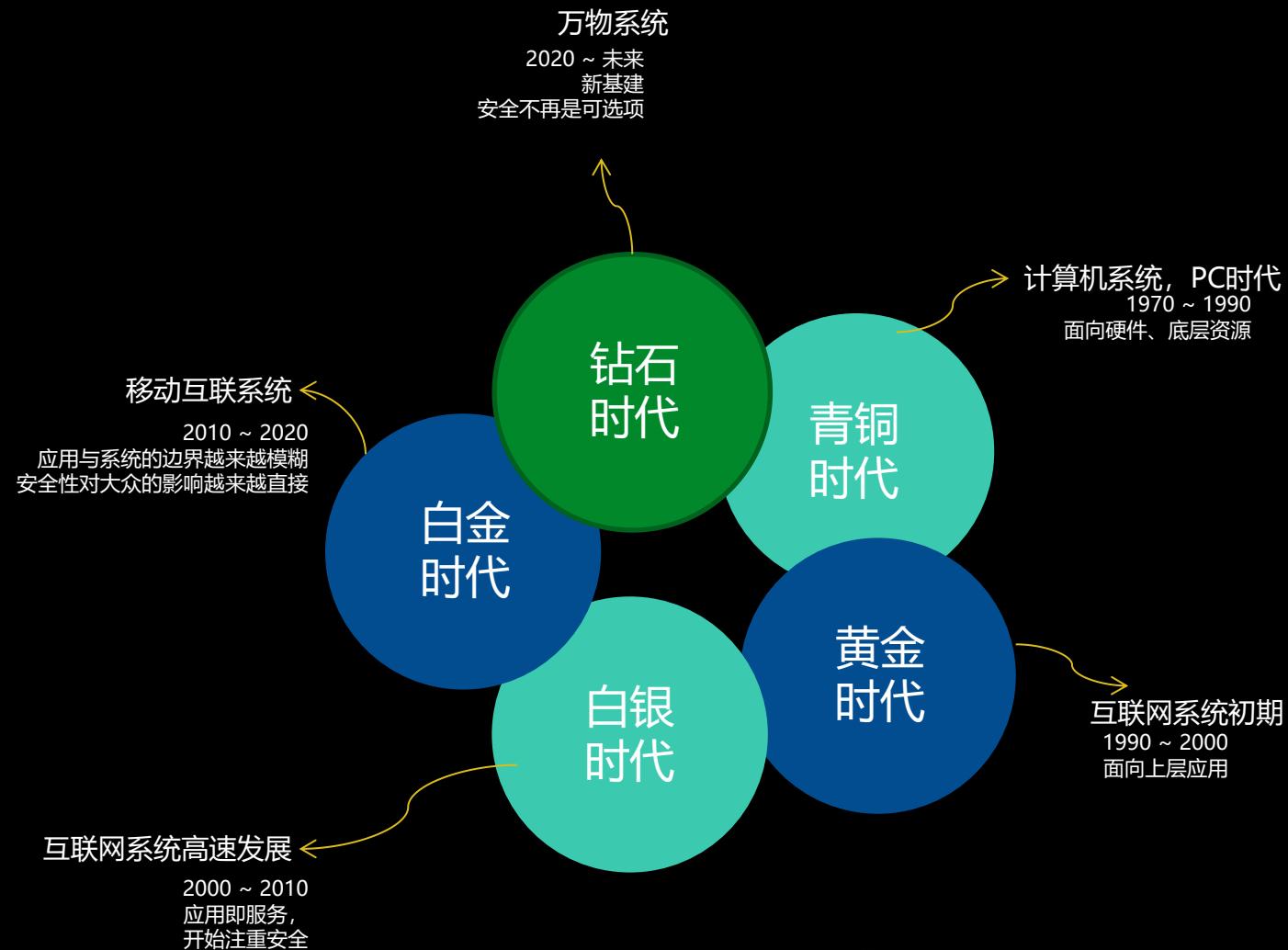
《Rust 编程之道》作者
企业独立咨询顾问
Rust 中文社区联合发起者

布道 Rust 过程中的小烦恼：

喜欢我的人调侃我 “Rust 之父”
不喜欢我的人喷我 “Rust 之父”

02

时代视角下 系统级开发



语言大师们的看法



LangNext 2014 (C++, Rust, D, Go)

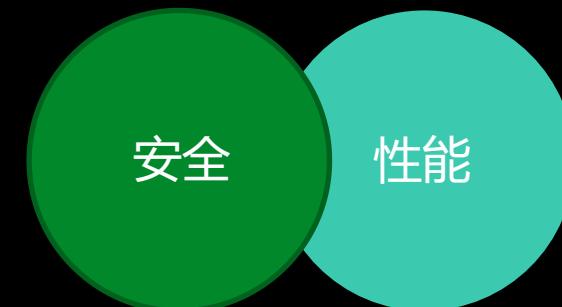
当下什么是系统级开发

能直接面向硬件，解决底层资源调度和性能问题

能保证基本的安全性，尽最大可能避免系统安全漏洞

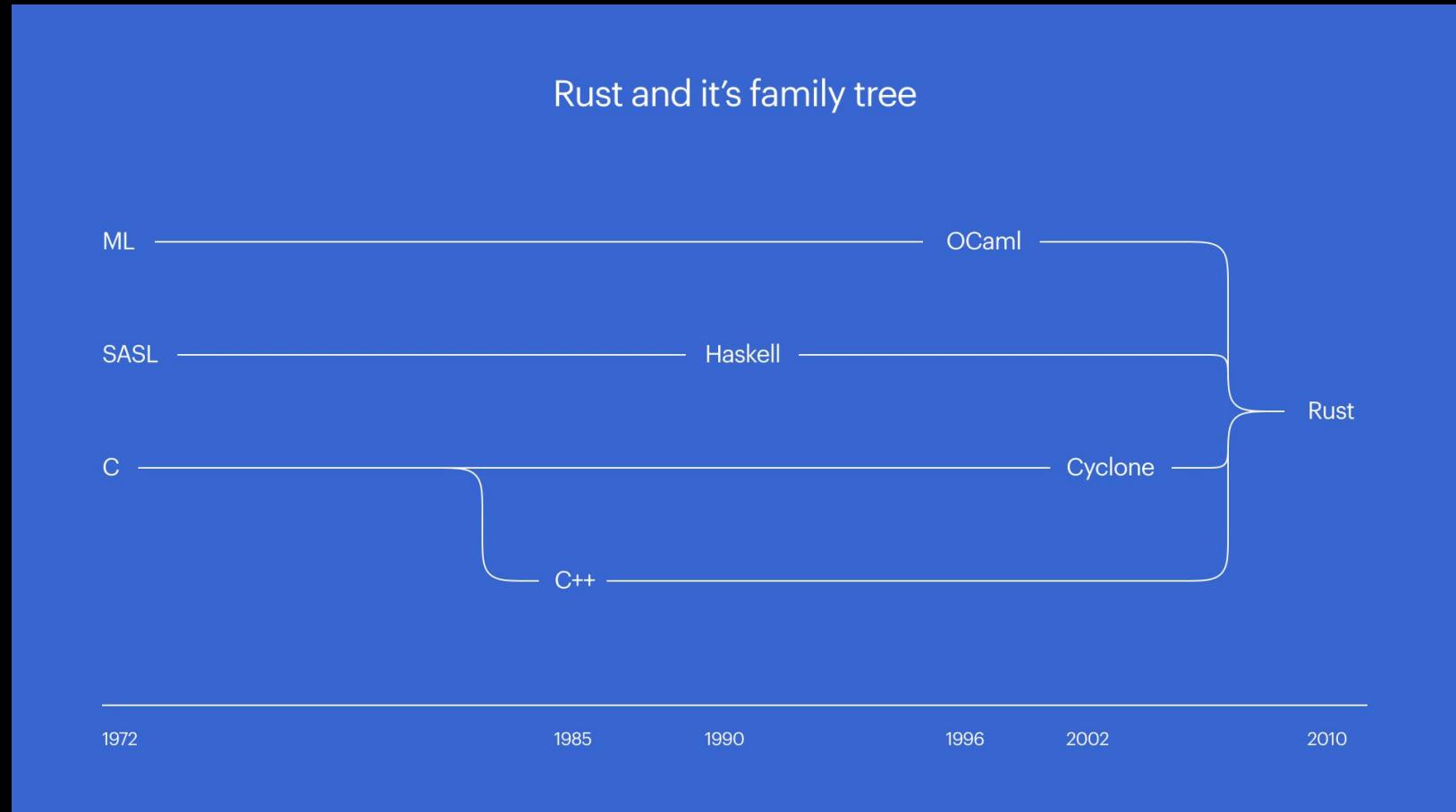
能保证系统的健壮性，保证程序中的错误都得到合理的处理

能保证系统灵活的可扩展性和易维护性，保证系统可以长期稳定提供服务



03

Rust 语言 的兴起



04

Rust 语言 的优势与劣势

1

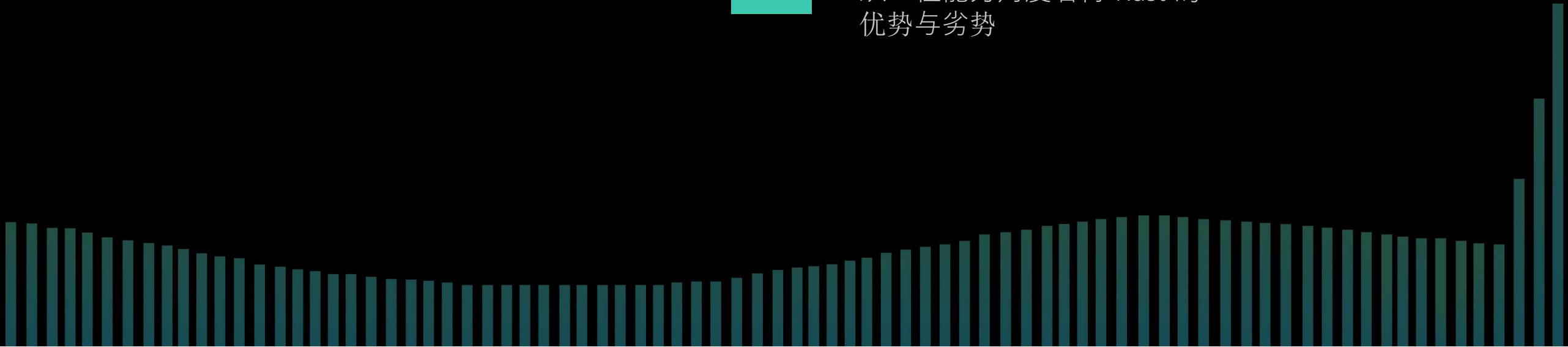
语言设计

从语言设计角度看待 Rust 的
优势与劣势

2

工程能力

从工程能力角度看待 Rust 的
优势与劣势



语言设计

1

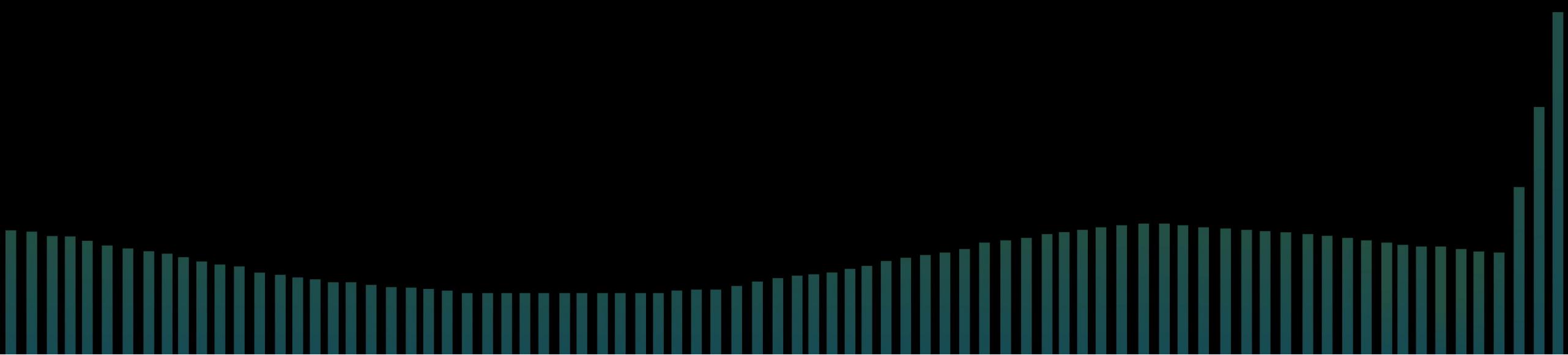
类型系统

虽然类型安全，但类型系统
还不够完善

2

安全模型

所有权机制保证安全，但带
来心智负担



语言设计

类型系统·角色

所有权语义

工程能力

类型系统

资源管理

语言设计

类型系统·优势

一切皆类型

简洁的内核：类型 + 行为

强大的静态类型检查

支持零成本抽象

基于类型的多范式特性

```
○○○

fn main(){
    let s = "abc?d";

    let mut chars = s.chars().collect::<Vec<char>>();

    for (i, c) in chars.iter_mut().enumerate() {
        let mut words = ('a'..='z').into_iter();

        if chars[i] == '?' {
            let left = if i==0 {None} else { Some(chars[i-1]) };
            let right = if i==s.len()-1 {None} else {Some(chars[i+1])};

            chars[i] = words.find(
                |&w| Some(w) != left && Some(w) != right
            ).unwrap();
        }
    }

    let s = chars.into_iter().collect::<String>();
    println!("{}:?", s);
}
```

语言设计

类型系统·劣势

高阶类型不支持，
表达力不足

不支持泛型特化

CTFE 支持不够完善

```
○○○

#![feature(specialization)]

trait Count {
    fn count(self) -> usize;
}

impl<T> Count for T {
    default fn count(self) -> usize {
        1
    }
}

impl<T> Count for T
where
    T: IntoIterator,
    T::Item: Count,
{
    fn count(self) -> usize {
        let i = self.into_iter();
        i.map(|x| x.count()).sum()
    }
}

fn main() {
    let v = vec![1, 2, 3];
    assert_eq!(v.count(), 3);

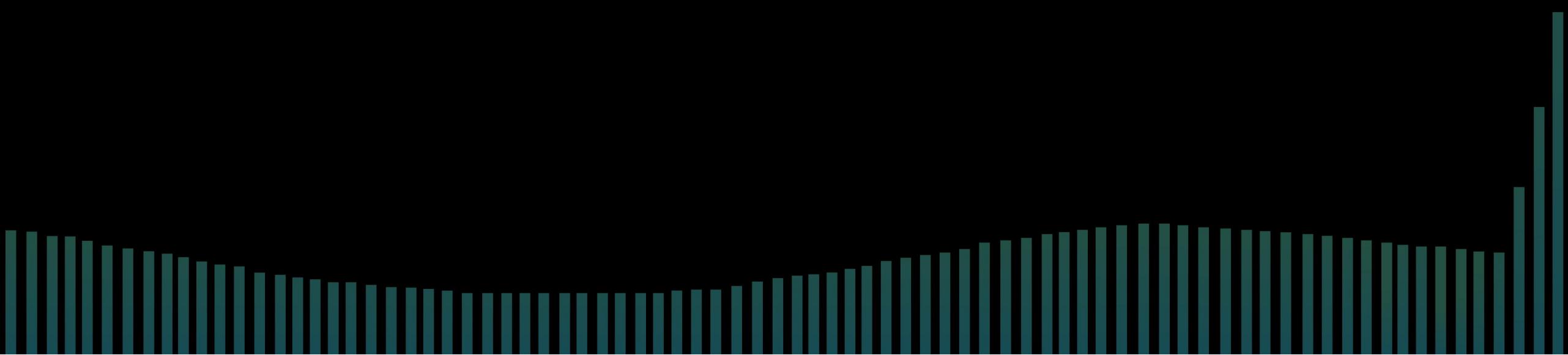
    let v = vec![vec![1, 2, 3], vec![4, 5, 6]];
    assert_eq!(v.count(), 6);
}
```

语言设计

安全模型·优点

高度一致性的所有权语义模型

Safe 与 Unsafe 界限分明



语言设计

安全模型•优点

高度一致性的所有权语义模型

Safe 与 Unsafe 界限分明

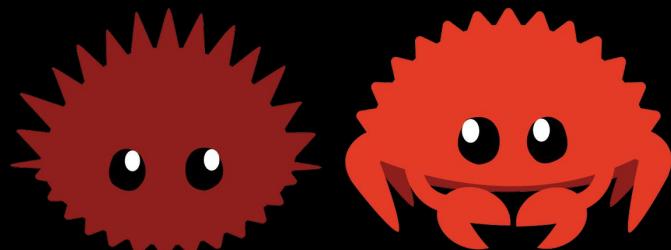
```
use std::thread;
fn main() {
    let mut s = "Hello".to_string();
    for _ in 0..3 {
        thread::spawn(move || {
            s.push_str(" Rust!");
        });
    }
}
```

语言设计

安全模型·优点

高度一致性的所有权语义模型

Safe 与 Unsafe 界限分明



```
○ ○ ○

) {

/// Safety:
/// 小心传入的 index 超过 数组长度导致数组越界
pub unsafe fn insert(&mut self, index: usize, element: T) {
    let len = self.len();

    // 通过判断长度是否达到容量极限来决定是否进行扩容
    if len == self.buf.cap() {
        self.reserve(1);
    }
    unsafe {
        {
            let p = self.as_mut_ptr().offset(index as isize);
            ptr::copy(p, p.offset(1), len - index);
            ptr::write(p, element);
        }
        self.set_len(len + 1);
    }
}
```

语言设计

安全模型•劣势

学习曲线高

开发者会有心智负担

```
fn main() {
    let v = vec![1,2,3, 4, 5,6];
    let mut buf = Buffer::new(&v);
    let b1 = buf.read_bytes();
    let b2 = buf.read_bytes();
    print(b1,b2)
}
fn print(b1 :&[u8], b2: &[u8]) {
    println!("{}:{} {}:{}", b1, b2)
}
struct Buffer<'a> {
    buf: &'a [u8],
    pos: usize,
}
impl<'a> Buffer<'a> {
    fn new(b: &'a [u8]) -> Buffer {
        Buffer {
            buf: b,
            pos: 0,
        }
    }
    fn read_bytes(&mut self) -> &'a [u8] {
        self.pos += 3;
        &self.buf[self.pos-3..self.pos]
    }
}
```

&'a [u8] {

]

t work

工程能力

可扩展性和易维护性

1

Rust 语言天生具备可扩展性和易维护性

健壮性

2

Rust 具有优雅的错误处理方式

简洁的抽象方式

3

Rust 是混合范式语言，但抽象方式和 C 语言一样简洁

现代化工具链

4

Rust 现代化的工具链，就是为工程而生

工程能力

可扩展性和易维护性

Rust 天生面向接口编程

```
enum Knob {
    Linear(LinearKnob),
    Logarithmic(LogarithmicKnob),
}

impl KnobControl for Knob {
    fn set_position(&mut self, value: f64) {
        match self {
            Knob::Linear(inner_knob) => inner_knob.set_position(value),
            Knob::Logarithmic(inner_knob) => inner_knob.set_position(value),
        }
    }

    fn get_value(&self) -> f64 {
        match self {
            Knob::Linear(inner_knob) => inner_knob.get_value(),
            Knob::Logarithmic(inner_knob) => inner_knob.get_value(),
        }
    }
}

//use the knobs
```

工程能力

可扩展性和易维护性

Rust 天生面向接口编程

显式哲学，几乎无隐式行为

```
trait KnobControl {
    fn set_position(&mut self, value: f64);
    fn get_value(&self) -> f64;
}

struct LinearKnob {
    position: f64,
}

struct LogarithmicKnob {
    position: f64,
}

impl KnobControl for LinearKnob {
    fn set_position(&mut self, value: f64) {
        self.position = value;
    }

    fn get_value(&self) -> f64 {
        self.position
    }
}

impl KnobControl for LogarithmicKnob {
    fn set_position(&mut self, value: f64) {
        self.position = value;
    }

    fn get_value(&self) -> f64 {
        (self.position + 1.).log2()
    }
}

fn main() {
    let v: Vec<Box<dyn KnobControl>> = vec![
        //set the knobs
    ];
    //use the knobs
}
```

工程能力

健壮性

失败 (Failure)

错误 (Error)

恐慌 (Panic)

```
use std::panic;

fn main() {
    let result = panic::catch_unwind(
        || { println!("hello!"); });
    assert!(result.is_ok());
    let result = panic::catch_unwind(
        || { panic!("oh no!"); });
    assert!(result.is_err());
    println!("{}", sum(1, 2));
}
```

工程能力

简洁的抽象方式

基于类型设计，OOP和FP只是语言特性，不需要纠结用哪个编程范式

```
trait Colorize {
    fn red(self) -> ColoredString;
    fn on_yellow(self) -> ColoredString;
}

impl<'a> Colorize for ColoredString {
    fn red(self) -> ColoredString {
        ColoredString{ fgcolor: String::from("31"), ..self }
    }
    fn on_yellow(self) -> ColoredString {
        ColoredString { bgcolor: String::from("43"), ..self }
    }
}

impl<'a> Colorize for &'a str {
    fn red(self) -> ColoredString {
        ColoredString {
            fgcolor: String::from("31"),
            input: String::from(self),
            ..ColoredString::default()
        }
    }
    fn on_yellow(self) -> ColoredString {
        ColoredString {
            bgcolor: String::from("43"),
            input: String::from(self),
            ..ColoredString::default()
        }
    }
}

impl fmt::Display for ColoredString {
    fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
        let mut input = &self.input.clone();
        try!(f.write_str(&self.compute_style()));
        try!(f.write_str(input));
        try!(f.write_str("\x1B[0m"));
        Ok(())
    }
}

fn main() {
    let hi = "Hello".red().on_yellow();
    println!("{}", hi);
    let hi = "Hello".on_yellow();
    println!("{}", hi);
    let hi = "Hello".red();
    println!("{}", hi);
    let hi = "Hello".on_yellow().red();
    println!("{}", hi);
}
```

工程能力

简洁的抽象方式

基于类型设计，OOP和FP只是语言特性，不需要纠结用哪个编程范式

```
-     let mut started = false;
-
+
+     let mut ssload = State::init();
+
+
+     if let Some(_) = self.get_start_func()?.is_some() {
+         self.restore_heap()?;
+         started = true;
+         ssload.start();
+     } else {
+         self.state = State::NotStarted;
+     }
@@
-
+
+     if self.get_start_func()?.is_some() && started == false {
+         self.state = State::NotStarted;
+     } else {
+         self.state = State::Ready;
+     }
-
```

工程能力

现代化工具链

强大的 Cargo 包管理器与模块化支持

高质量的第三方库

- **Cargo Clippy**
- **Cargo Audit**
- **Cargo Deny**
- **Cargo Outdated**
- **Cargo Expand**
- **Cargo Bloat**

工程能力

现代化工具链

强大的 Cargo 包管理器与模块化支持

高质量的第三方库(crates.io)

The screenshot shows the homepage of crates.io, a Rust package registry. At the top, there's a navigation bar with the logo, a search bar containing 'Click or press 'S' to search...', and links for 'Browse All Crates', 'Docs', 'Log in with GitHub', and 'Fork me on GitHub'. The main title is 'The Rust community's crate registry'. Below it are two buttons: 'Install Cargo' and 'Getting Started'. A central text block says: 'Instantly publish your crates and install them. Use the API to interact and find out more information about available crates. Become a contributor and enhance the site with your work.' To the right, there are two large statistics: '4,778,155,628 Downloads' with a download icon, and '50,553 Crates in stock' with a box icon. Below these are three sections: 'New Crates' (listing 'bevy_spirv v0.0.1' and 'bevy_rustgpu v0.0.1'), 'Most Downloaded' (listing 'rand' and 'syn'), and 'Just Updated' (listing 'opentelemetry-tide v0.5.0' and 'basic_lib_for_me v0.3.4'). The background features a dark teal gradient with vertical bars.

工程能力

劣势

编译速度慢

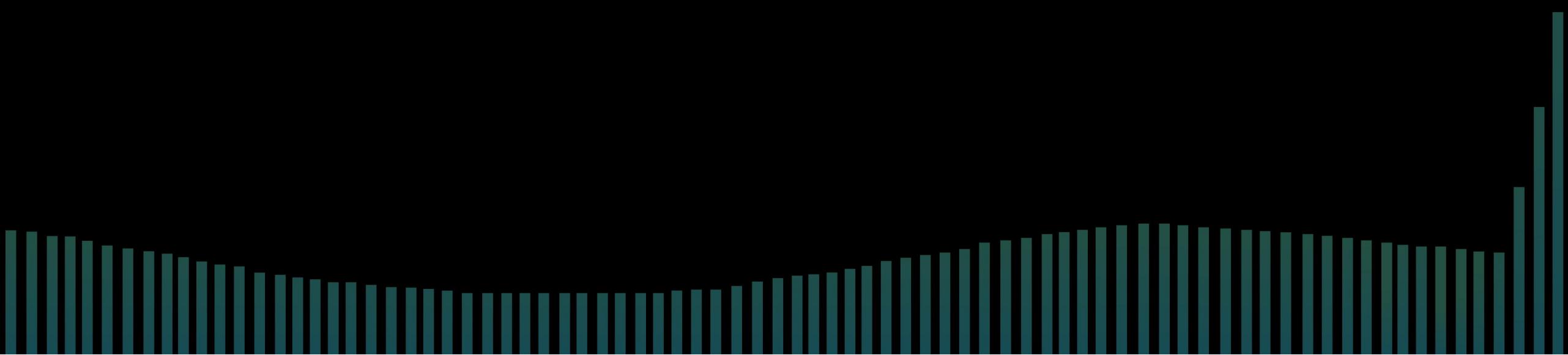
产出的文件尺寸较大

Unsafe Rust UB 检查不够完善

生态中开箱即用的库不多

IDE 支持不够完善

Rust 专用的调试工具不够完善



工程能力

没有展开提到的 Rust 其他优势

出众的元编程能力

跨平台支持

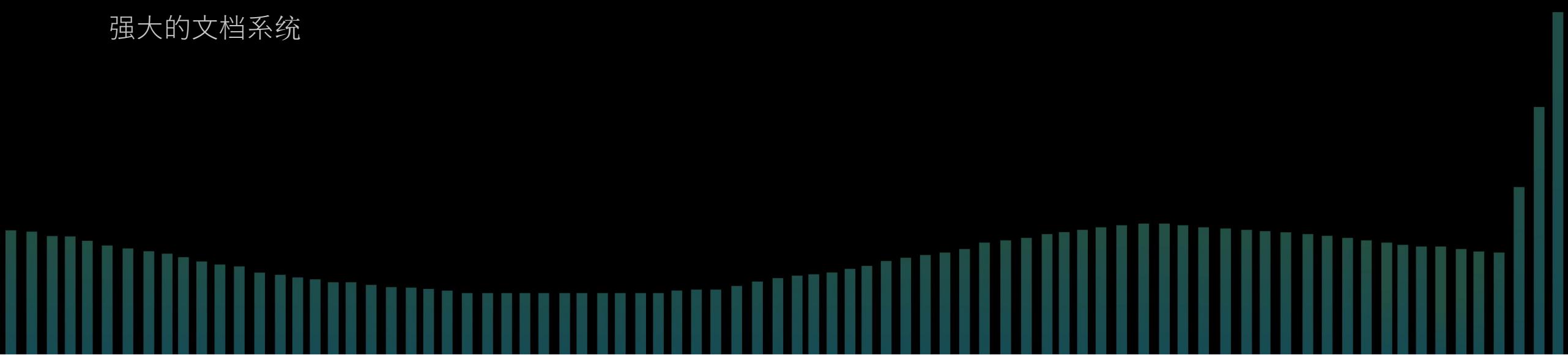
安全易用的异步编程模型

强大的文档系统

成熟的开源社区

无缝与C-ABI 接口调用

内建单元测试和性能测试支持



谢谢

张晓龙

中兴通讯资深架构师



目前就职于中兴通讯，资深软件架构师，高级技术教练，Go 语言 gomonkey 和 trans-dsl 作者，具有十多年软件架构与设计经验。近年来专注于 PaaS 和 5G 等大型平台的软件架构与设计，尤其对于领域驱动设计、微服务框架和契约测试有深入研究。曾指导多个团队积极实践 DDD（领域驱动设计），包括开发领域和测试领域，取得了很好的实践成果。

主办方：

Boolan
高端 IT 咨询与教育平台

CPP-Summit 2020

张晓龙
资深架构师

中兴契约测试规模 化落地实践

议程

1

契约测试介绍

契约测试的背景、概念和开源框架

2

中兴契约测试技术实践

契约测试框架 Coral（自研），契约测试用例开发套路，契约测试最佳实践

3

中兴契约测试工程管理实践

契约测试试点及推广，契约测试嵌入团队研发流程，契约测试成果和收益

01

契约测试介绍

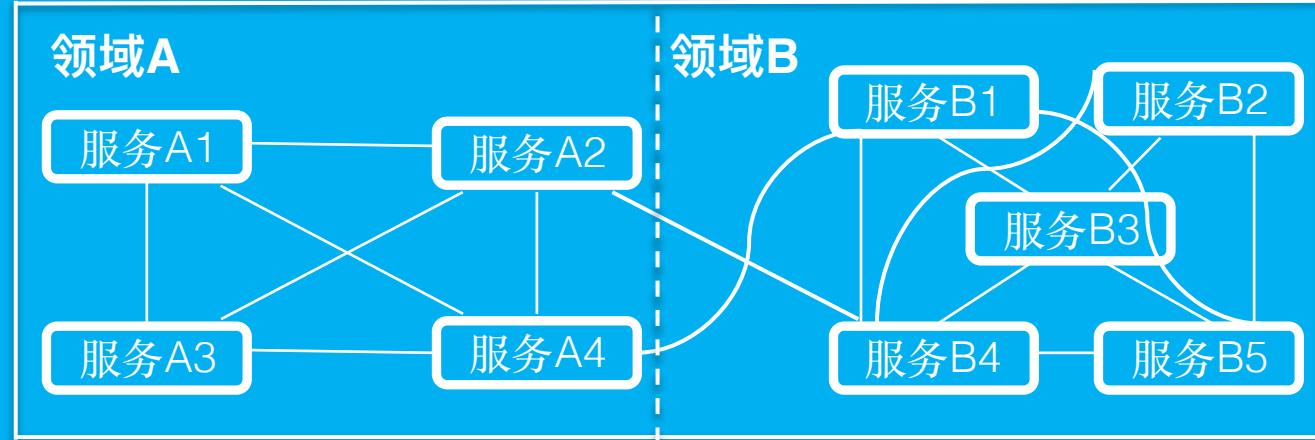
契约测试的背景，概念和开源框架

系统可靠性

线性系统

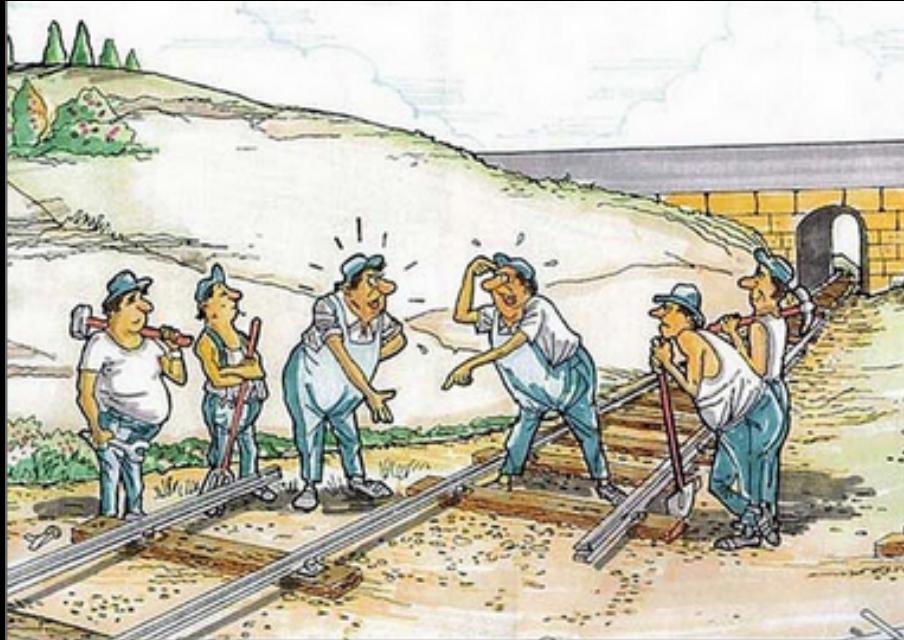


大型微服务系统



- 线性系统（即复杂性随规模呈线性增长）的可靠性等于组成它的各个服务的可靠性之乘积，即系统整体的可靠性 ($90\% \times 90\% \times 90\% \times 90\% = 65.61\%$) 低于任一服务的可靠性 (90%)
- 一个大型微服务系统，每一个服务的可靠性都对系统整体的可靠性有着非常重要的影响，同时各个服务之间的依赖关系也会对系统的可靠性产生显著影响

如何提高可靠性?



尽早联调

契约测试

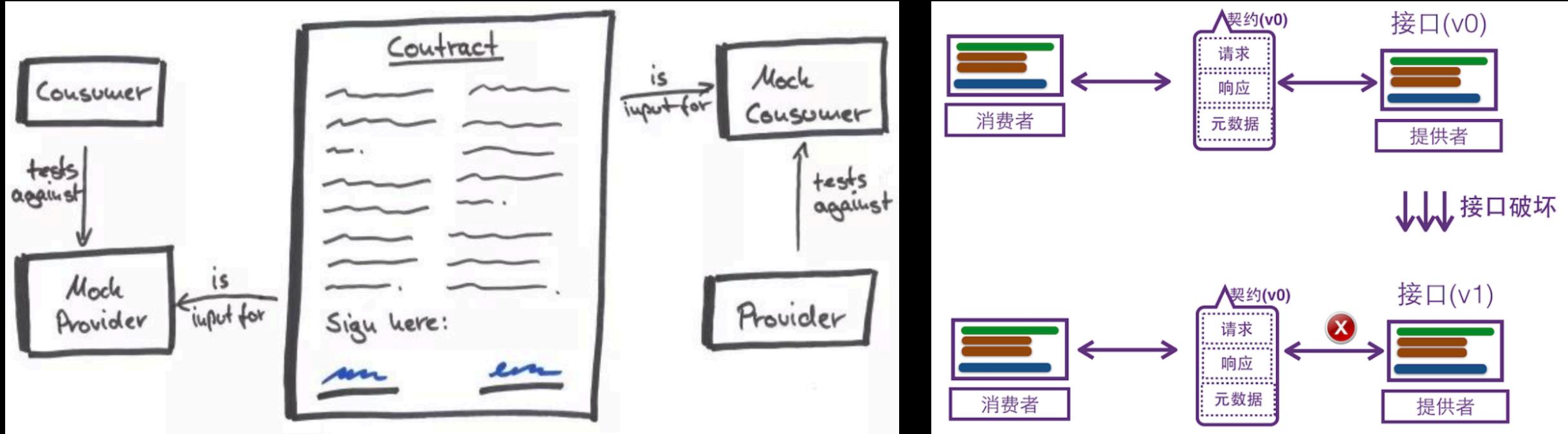
轻量化集成

契约



将契约文字刻写在器皿上（纸上），盖上章（按手印），就是为了使契文中规定的内容得到多方承认和信守

契约测试



- > Consumer 和 Provider 双方可以独立的进行预集成测试，将本来在联调中才能触发的问题前移
- > 契约测试守护接口的正确性，一旦接口被破坏，会快速反馈出来

开源契约测试框架

| 契约测试框架 | 说明 | 适用场景 |
|-----------------------|-----------------------------------|---------------------|
| Pact | 轻量级、支持多种语言应用、支持与 Maven/Gradle 等集成 | 使用非 Spring Cloud 框架 |
| Spring Cloud Contract | 支持基于 JVM 的应用、支持与 Spring 其它组件集成 | 使用 Spring Cloud 框架 |

Pact

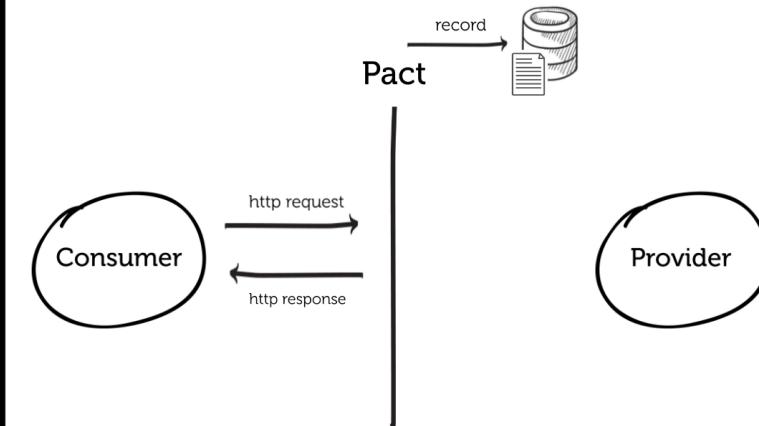
> 步骤 1：消费者

- 编写并运行单元测试（包括对接口的请求参数和预期响应）
- Pact 代替实际服务提供者（自动）
- 生成契约文件（自动）

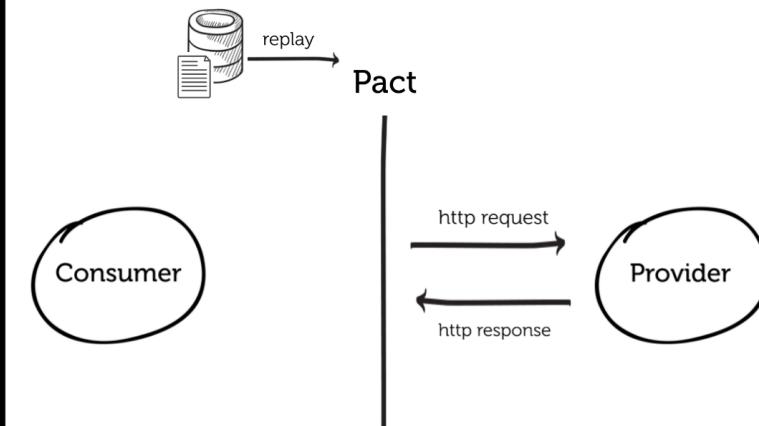
> 步骤 2：提供者

- 启动服务提供者
- Pact 回放契约文件中的请求，验证真实响应是否满足预期（自动）

Step 1: Define consumer expectations

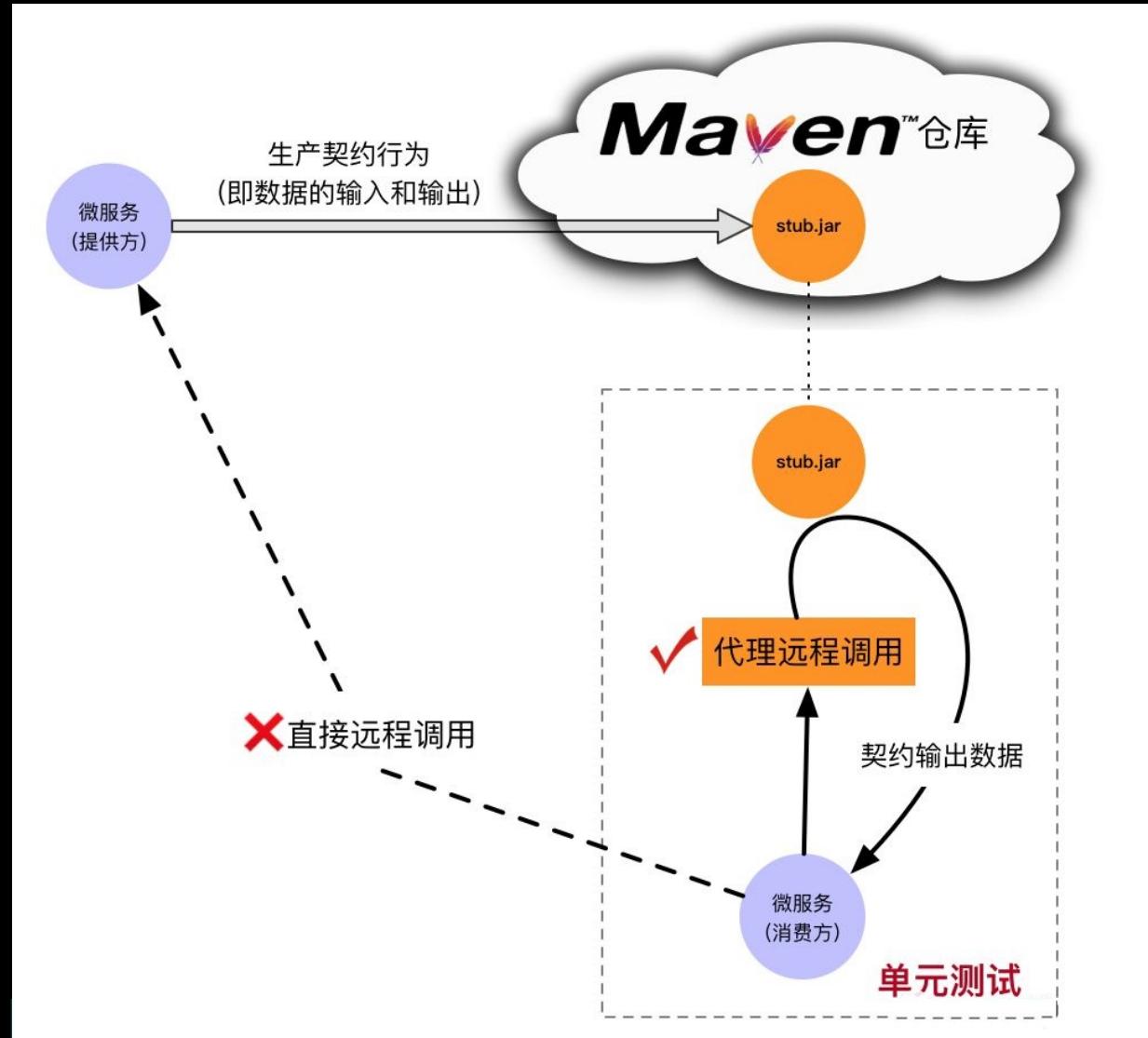


Step 2: Verify expectations on provider



Spring Cloud Contract

- 步骤 1：提供者
 - 编写契约
 - 编写测试基类，生成测试用例
 - 运行测试用例，验证真实响应是否满足预期（自动）
 - 发布 stub.jar 包
- 步骤 2：消费者
 - 编写测试用例
 - 通过注解指定依赖的 stub.jar 包
 - 运行测试用例，验证外部服务正常



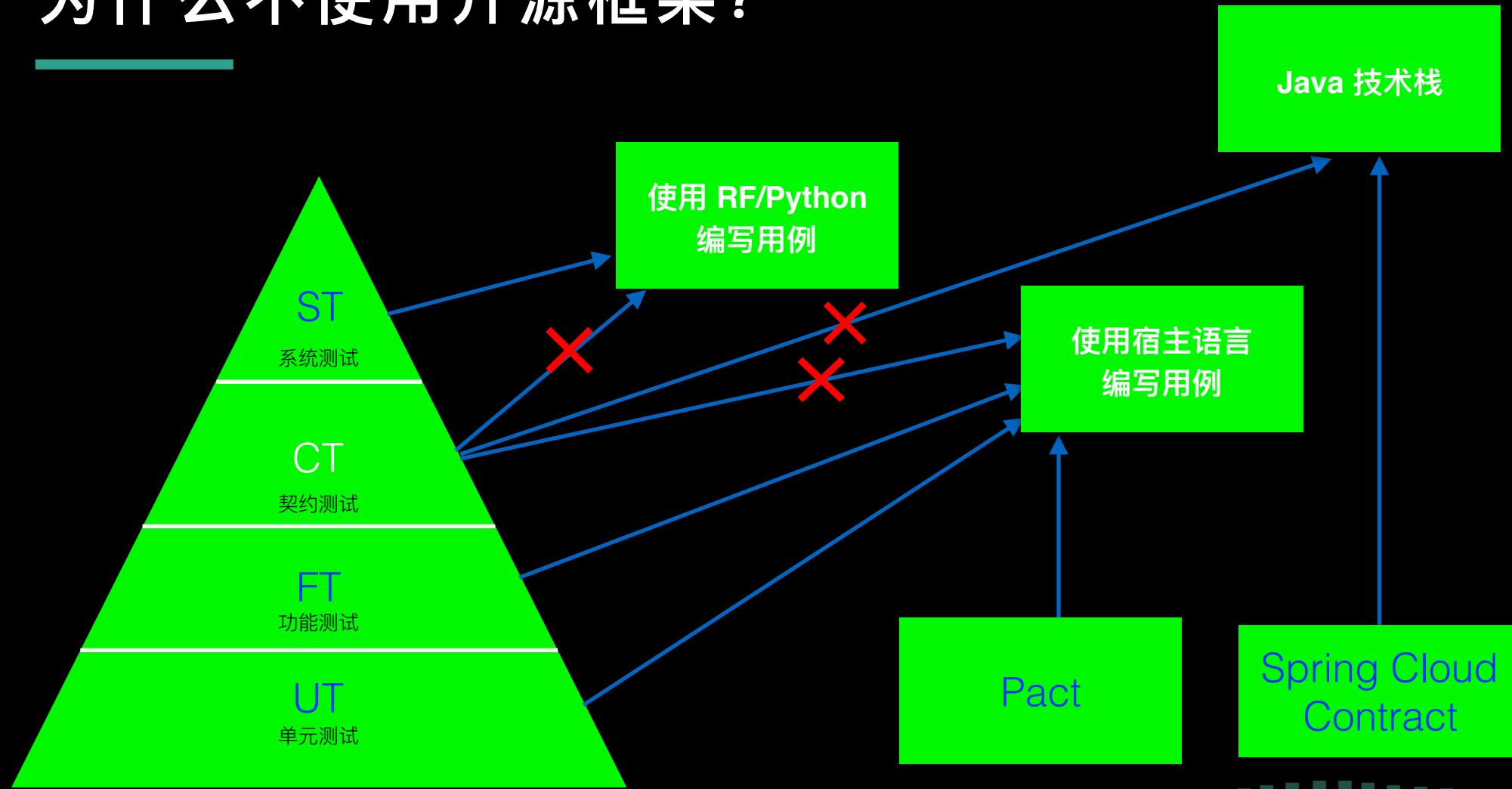
02

中兴契约测试技术实践

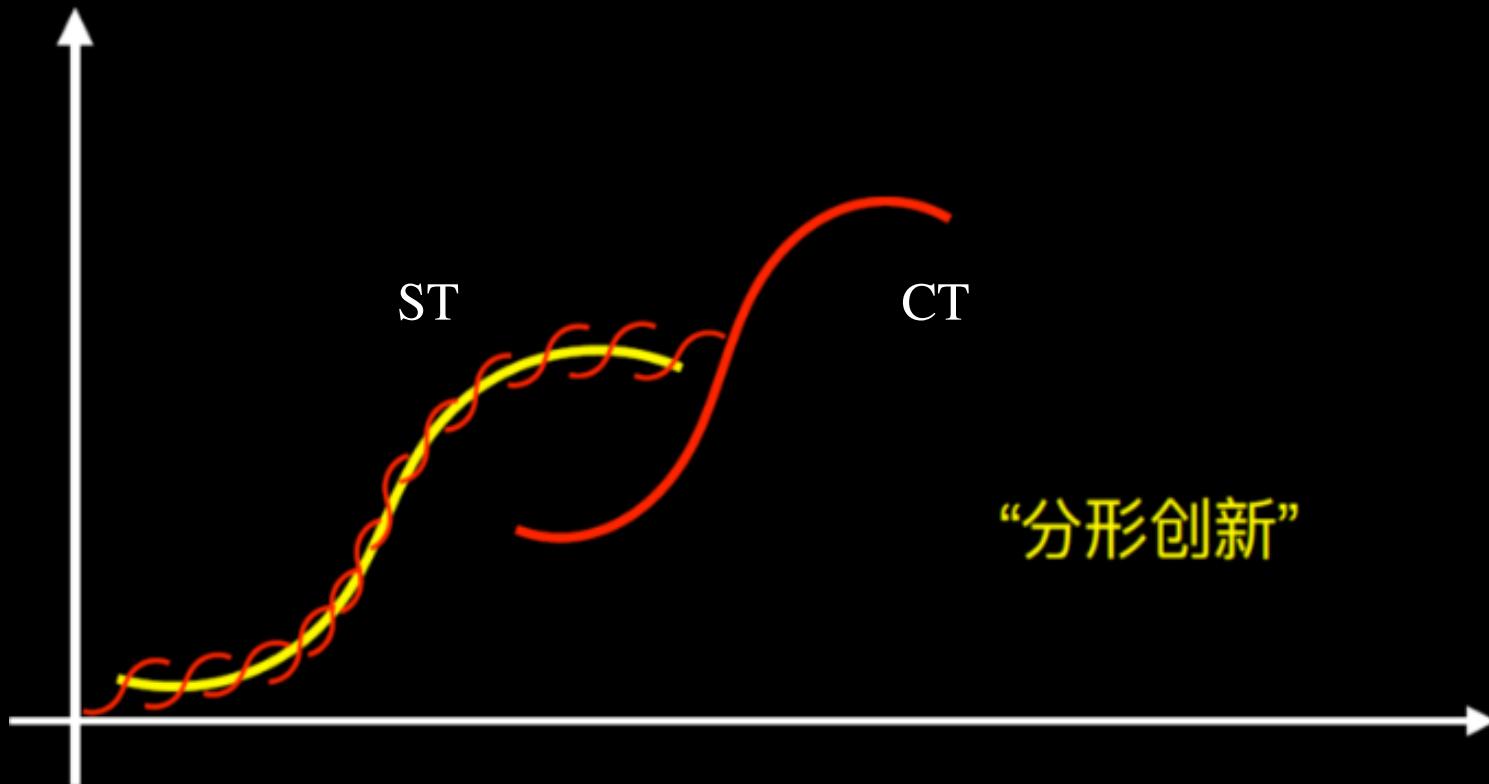
契约测试框架 Coral (自研) , 契约测试用例开发套路, 契约测试最佳实践

契约测试框架选型

为什么不使用开源框架？



契约测试的定位



领域诉求

- CT 从 ST 分形而来，SUT 的基础设施尽可能是真的
- 除过服务契约，还要有流程契约
- 无码化契约测试用例开发，使得懂点业务的同学就可以参与，具有普世价值
- CT 必须能在开发虚机部署运行，尽可能的轻量化

为什么不基于开源框架二次开发？

- 技术栈不同
- 能够直接复用的特性不多

Coral 契约测试介绍

核心设计：流程契约&流程契约实例

```
@startuml
title procedure.context
```

T1 -> S1: msg1
note left: "REST", "json"

T1 --> S2: msg2:request
note left: "REST", "json"

S2 -> N1: msg3
note right: "REST", "json"

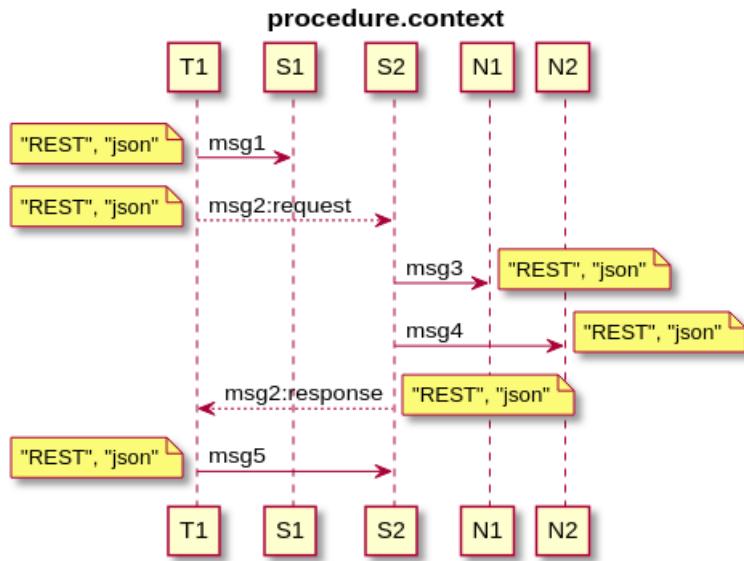
S2 -> N2: msg4
note right: "REST", "json"

S2 --> T1: msg2:response
note right: "REST", "json"

T1 -> S2: msg5
note left: "REST", "json"

```
@enduml
```

流程契约



```
@startuml
title procedure.context
```

FakeApp -> S1: msg1
note left: "REST", "json"

FakeApp --> S2: msg2:request
note left: "REST", "json"

S2 -> FakeApp: msg3
note right: "REST", "json"

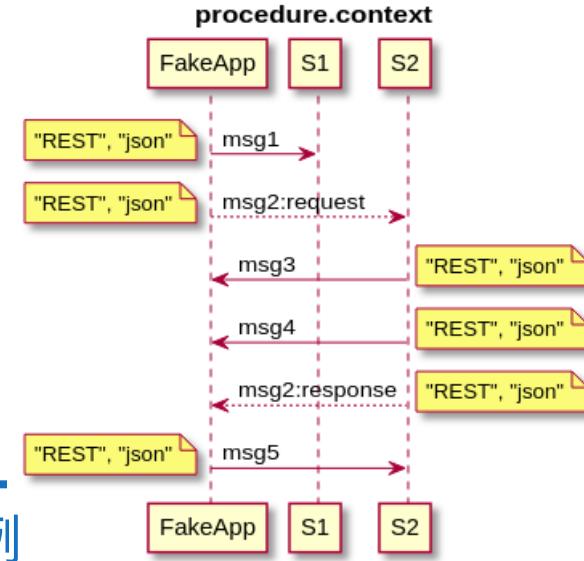
S2 -> FakeApp: msg4
note right: "REST", "json"

S2 --> FakeApp: msg2:response
note right: "REST", "json"

FakeApp -> S2: msg5
note left: "REST", "json"

```
@enduml
```

流程契约实例



核心设计：服务契约&服务契约实例

```
swagger: '2.0'  
info:  
  title: create_rcslist_succ.template  
  description: The service contract of the create_rcslist_succ msg  
  version: 1.0.0  
paths:  
  /api/v1/namespaces/{namespace}/rcslist:  
    post:  
      summary: create rc list of the namespace  
      description: create some rc objs in the namespace      parameters:  
        - name: namespace  
          in: path  
          description: name of the namespace  
          required: true  
          type: string  
        - name: body  
          in: body  
          description: rc list  
          required: true  
          schema:  
            $ref: '#/definitions/rclist'  
responses:  
  '200':  
    description: operate successfully  
    schema:  
      type: object  
      properties:  
        status:  
          type: string
```

服务契约

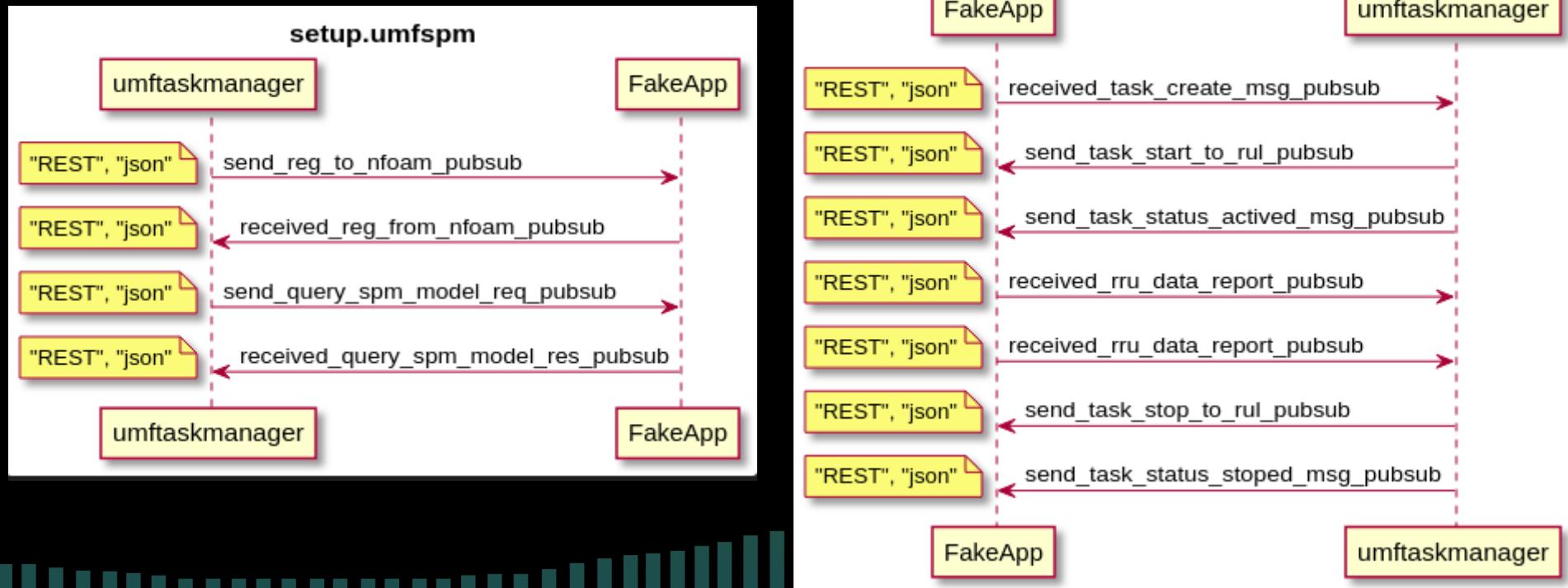


```
{  
  "base_info":{  
    "protocol":"http",  
    "port":"PROVIDER_PORT"  
  },  
  "request":{  
    "method":"GET",  
    "url":"/foobar",  
    "headers":{  
      "Content-Type":"application/json"  
    },  
    "body":{}  
  },  
  "response":{  
    "status": 200,  
    "headers": {  
      "Content-Type": "text/plain"  
    },  
    "body": {  
      "status": "${STATUS}"  
    }  
  },  
  "parameters": [  
    {  
      "name":"STATUS",  
      "value": "[foo,foo1,foo2]",  
      "default": "foo1"  
    }  
  ]  
}
```

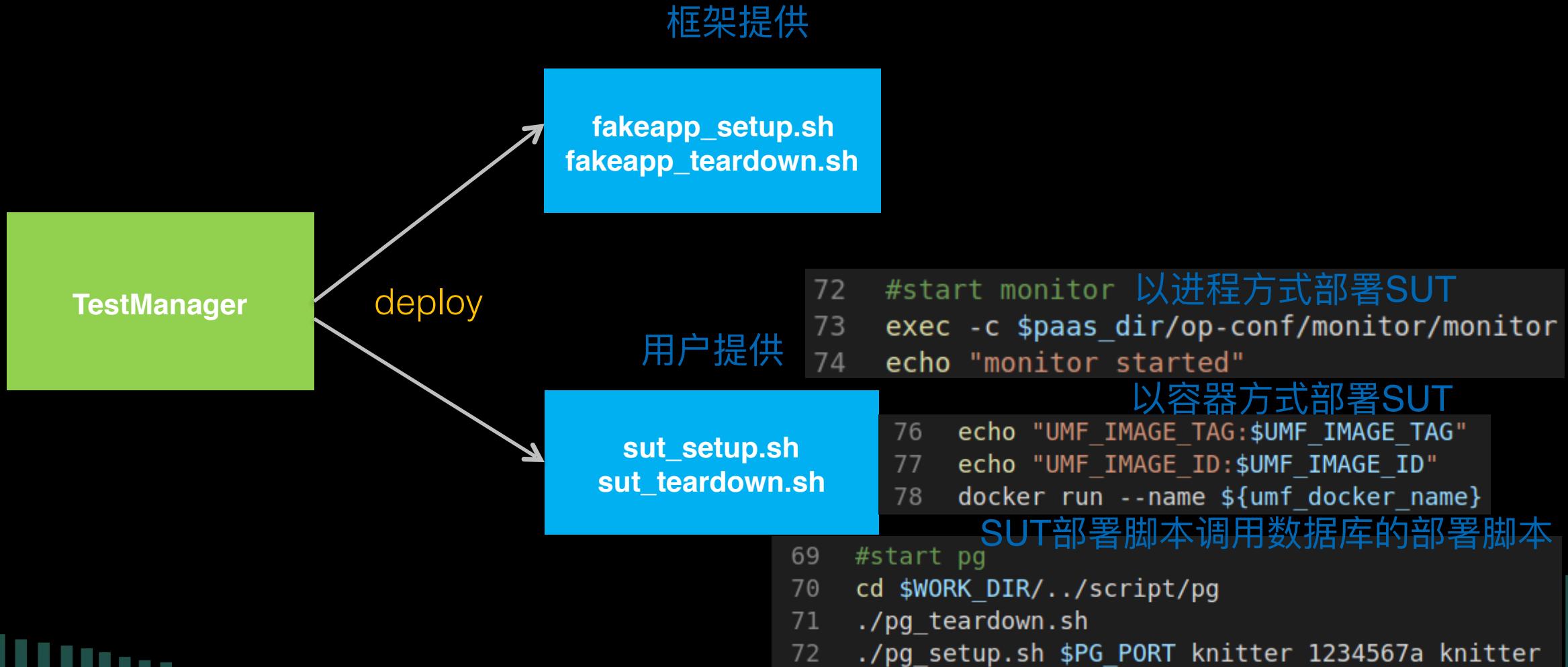
服务契约实例

核心设计：测试套件

- 根据四元组 (procedure, context, domain, tag_exp) 过滤用例集
- setup用例



核心设计：SUT 部署



核心设计：契约中心

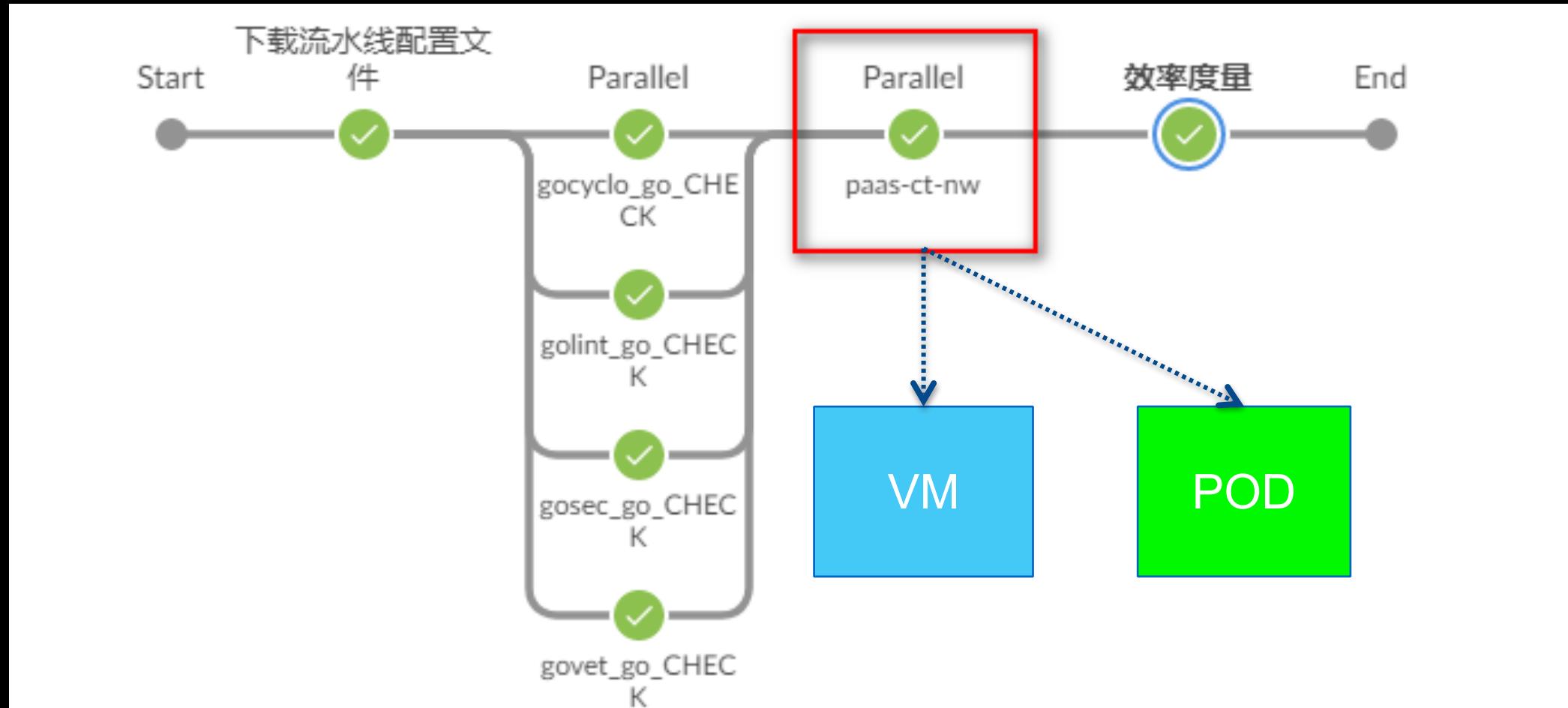
契约中心



- ◆ 服务契约

- ◆ 流程契约，可关联到服务契约库的服务契约
- ◆ 契约用例（流程契约实例+服务契约实例），
流程契约实例，可关联到服务契约实例
- ◆ SUT部署脚本

核心设计：CI 上线



核心设计：结构化日志

- TEST /home/coraldata/testcase/ops/monitor_commsrv/succ/inst/monitor.puml

Full Name: suites.monitor./home/coraldata/testcase/ops/monitor_commsrv/succ/inst/monitor.puml

Start / End / Elapsed: 20201113 14:08

Status: PASS (critical)

+ KEYWORD **get_clusters_info**

+ KEYWORD **get_nodes**

+ KEYWORD **watch_pods**

+ KEYWORD **commsrv_notify**

+ KEYWORD **get_commsrv_topo**

+ KEYWORD **query_commsrv_monitor**

```

sequenceDiagram
    participant Monitor
    participant FakeApp
    participant Opslet
    Note over Monitor, FakeApp, Opslet: monitor_commsrv.succ
    Monitor->>FakeApp: "REST", "json", "replace_port"
    Note over "REST", "json", "replace_port": "get_clusters_info"
    Monitor->>FakeApp: "REST", "json"
    Note over "REST", "json": "get_nodes"
    Monitor->>FakeApp: "REST", "json"
    Note over "REST", "json": "watch_pods"
    FakeApp-->>Monitor: "REST", "json"
    Note over "REST", "json": "commsrv_notify"
    Monitor->>Opslet: "REST", "json"
    Note over "REST", "json": "get_commsrv_topo"
    Opslet-->>Monitor: "REST", "json", "delay_req"
    Note over "REST", "json", "delay_req": "query_commsrv_monitor"
  
```

契约测试用例开发步骤

- 1. 编写流程契约
- 2. 编写服务契约
- 3. 生成流程契约实例
- 4. 构造服务契约实例
- 5. 编写 SUT 部署脚本 (Once)
- 6. 调试
- 7. 上传契约中心
- 8. 上线 CI

Coral 契约测试框架介绍

契约测试框架 Coral



两个角色



三方关系

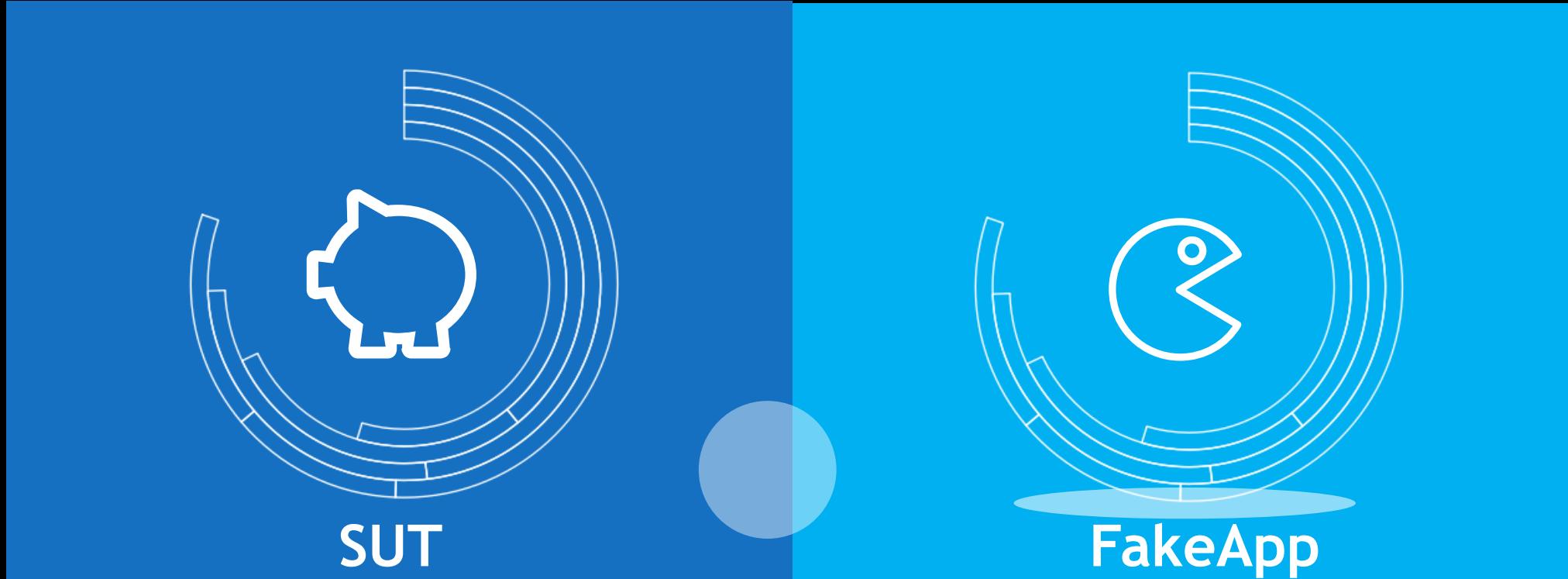


四项特征



八大价值

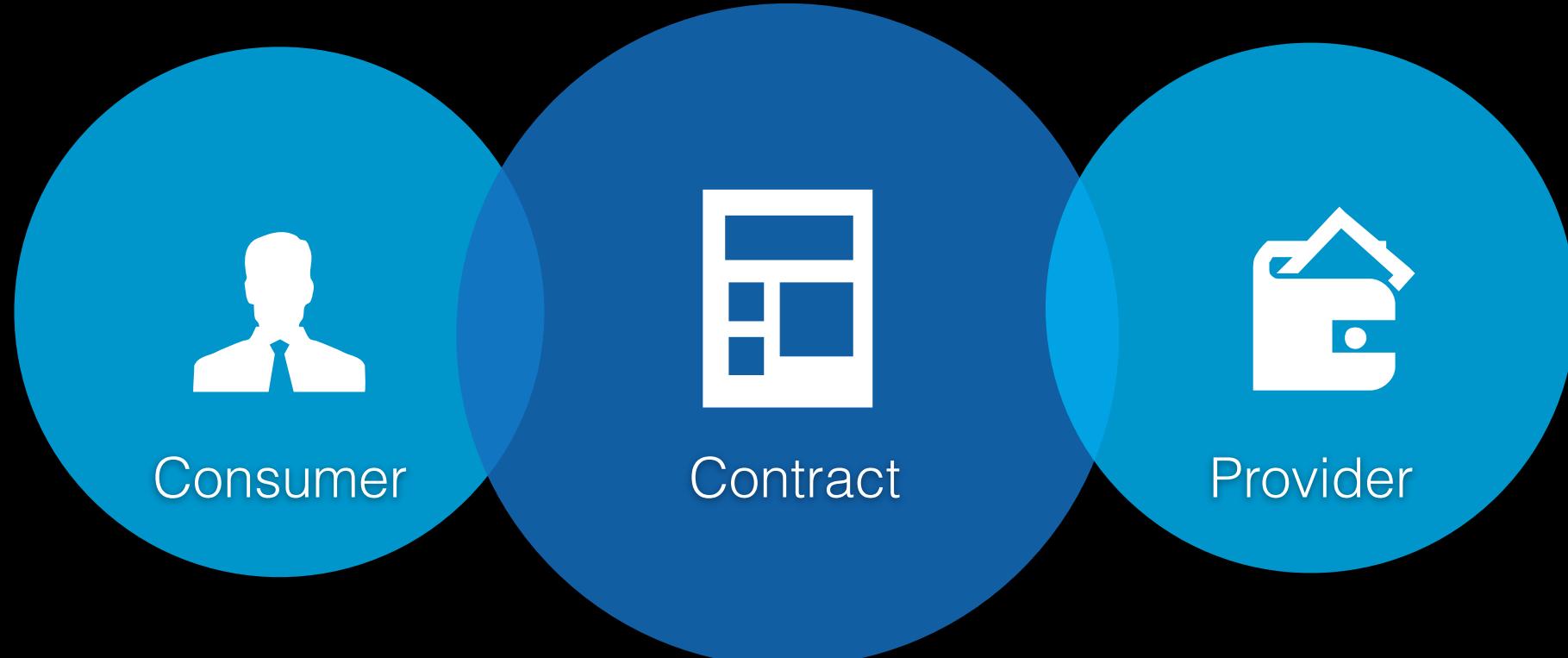
两个角色



被测系统（System Under Test）：被测系统是当前被测试的系统，根据测试系统不同，SUT 所指代的内容也不同，可以是一个类也可以是整个系统

桩应用：测试中通过 Mock 实现的服务，除过 SUT 之外的所有服务

三方关系



- Consumer 为服务消费者, Provider 为服务提供者, Contract 为两者之间交互的契约
- Consumer 不再依赖 Provider, 而是两者都依赖于 Contract

四项特征



轻量级

服务拆分
按需组合

通用性

为 ICT 而生
微服务架构

易用性

一键部署
一键测试

标准化

紧靠业界标准
没有特殊处理

八大价值

低成本回放系统测试

不用搭建系统测试环境
部署便捷
测试运行时间短



增加防火墙厚度

异常容易制造
用例覆盖面广



联调双方解耦

联调双方都依赖于契约
契约测试用例提前上 CI
降低联调成本



测试分层更合理

处于 FT 和 ST 之间专注领域内
ST 变薄专注领域间
用例维护成本降低



TS 根据三原则给用例打标签：场景覆盖到、成本低、最小知识

回放产品用户契约

录制产品用户契约
版本发布前在真实环境部署
自动回放用户契约以确保版本质量



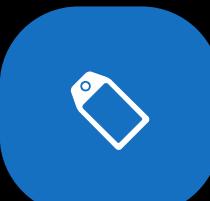
用例 CI 与文档 CI

用例 CI 守护用例质量
文档 CI 自动生成 API 文档
版本、用例和文档一致



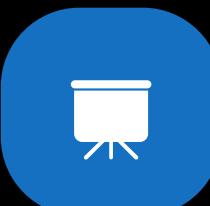
活文档

显式化领域边界行为
文档与代码同步变更
学习领域知识最生动的教材

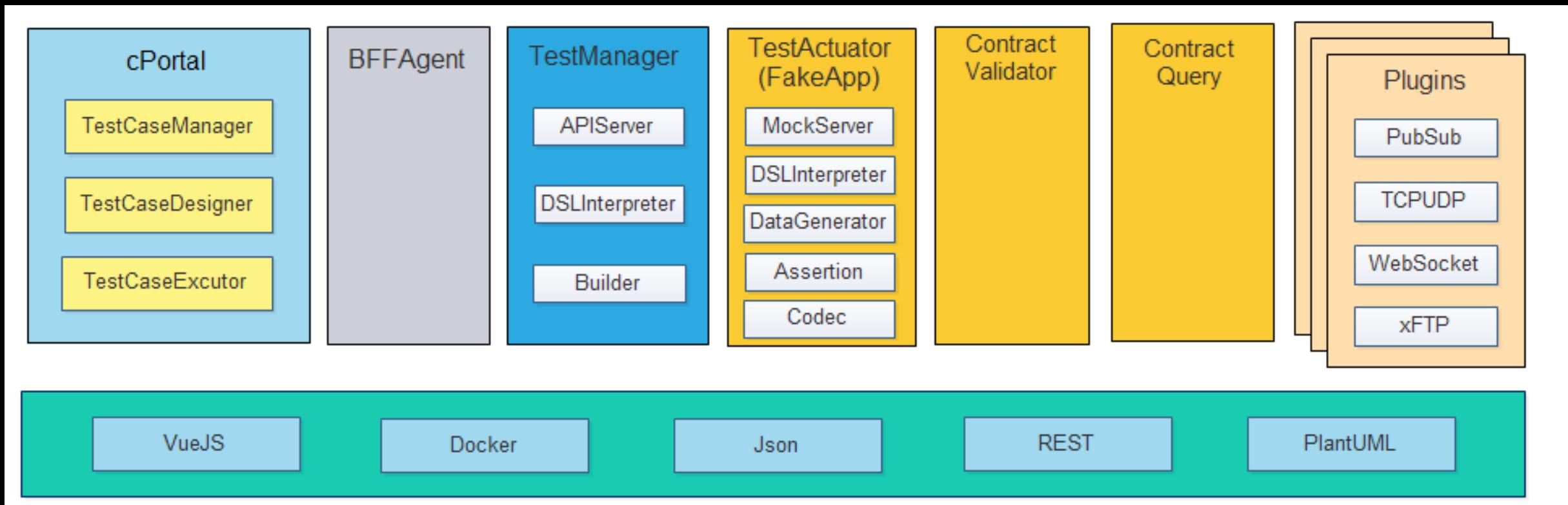


增加接口变动的安全性和准确性

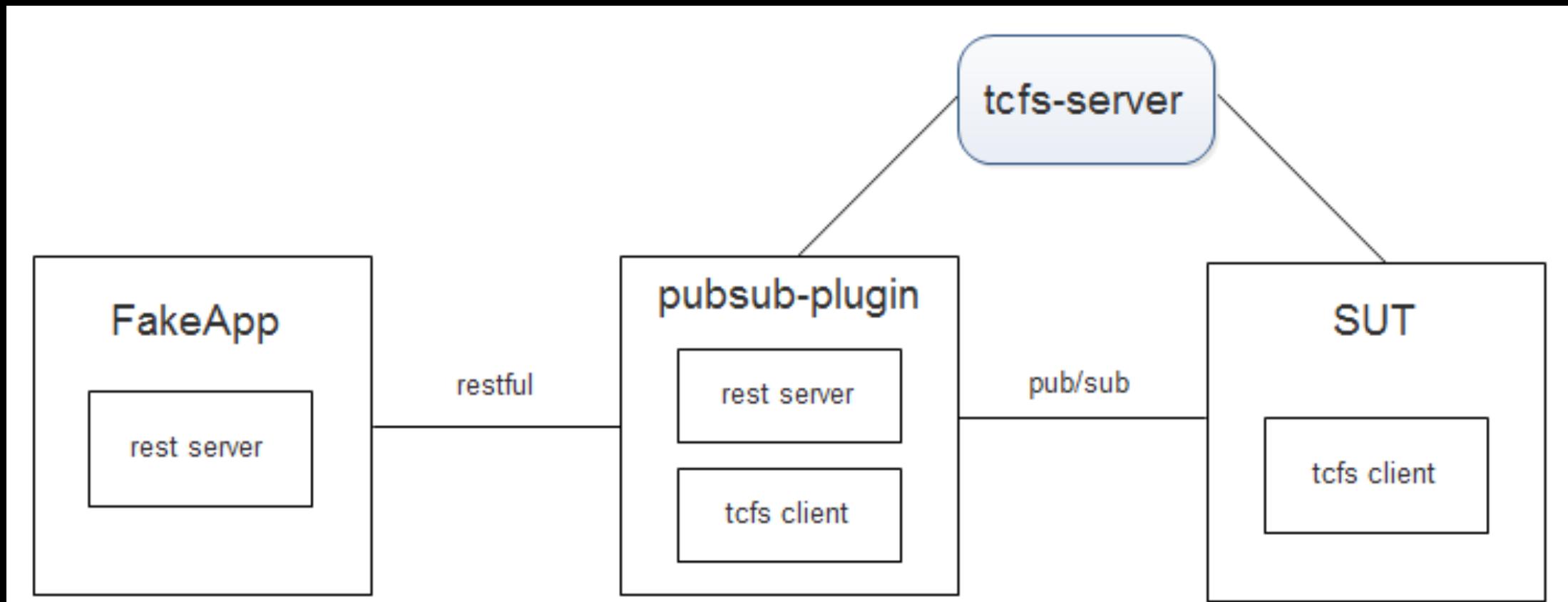
接口设计约束
接口变动监控
降低验证成本



Coral 微服务架构



插件设计



Coral 契约测试最佳实践

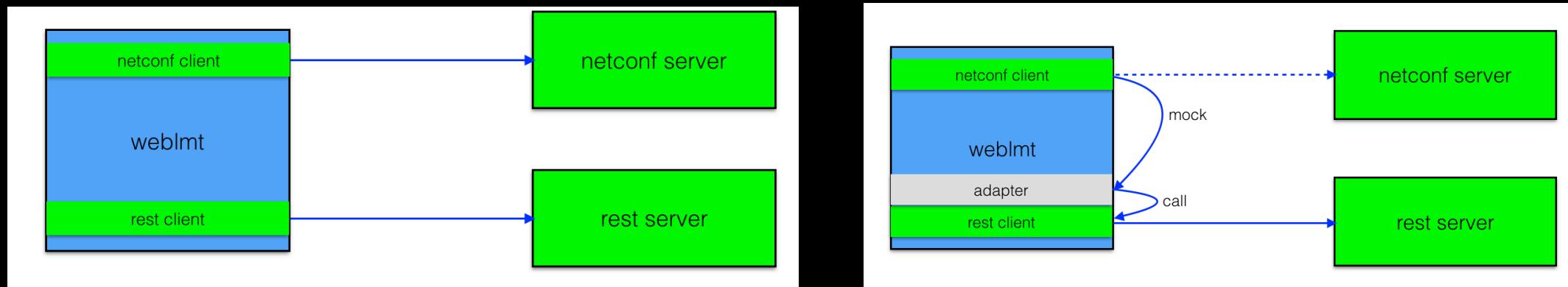
最佳实践一：NETCONF 协议

挑战

- SUT 依赖的协议重，模拟成本太高，真实部署太耗资源

解决方案

- 当 CT 模式时偷梁换柱（SUT Mock）



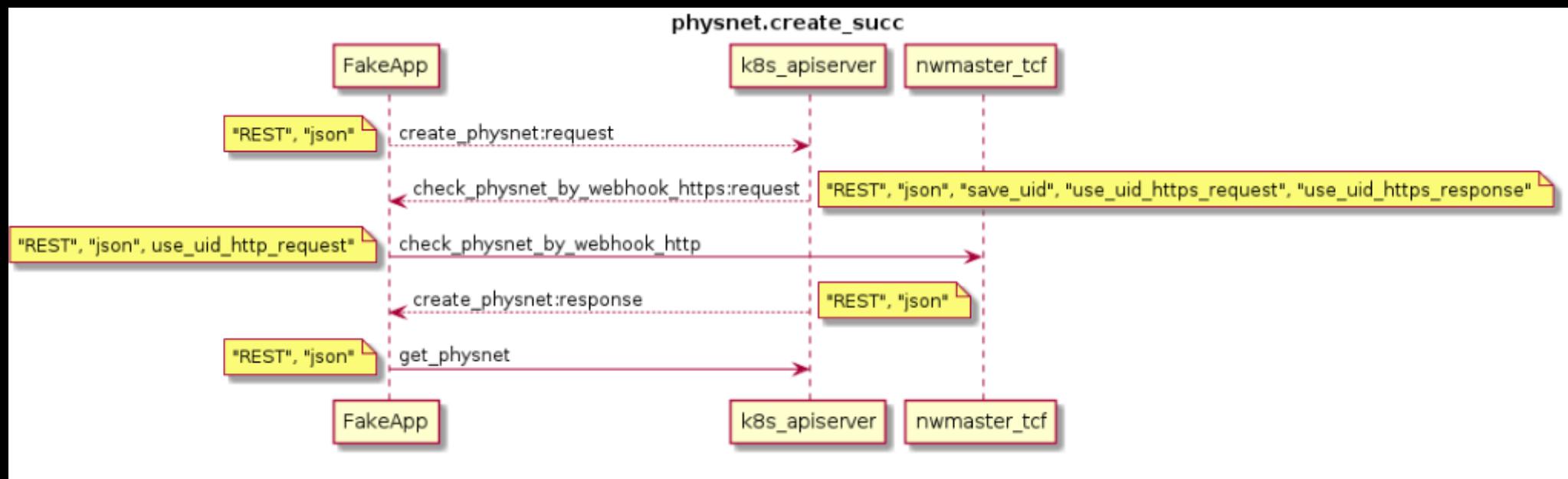
最佳实践二：K8S

挑战

> K8S 行为比较复杂，模拟成本太高

解决方案

> 将 APIServer 和 ETCD 作为 SUT 的一部分部署起来



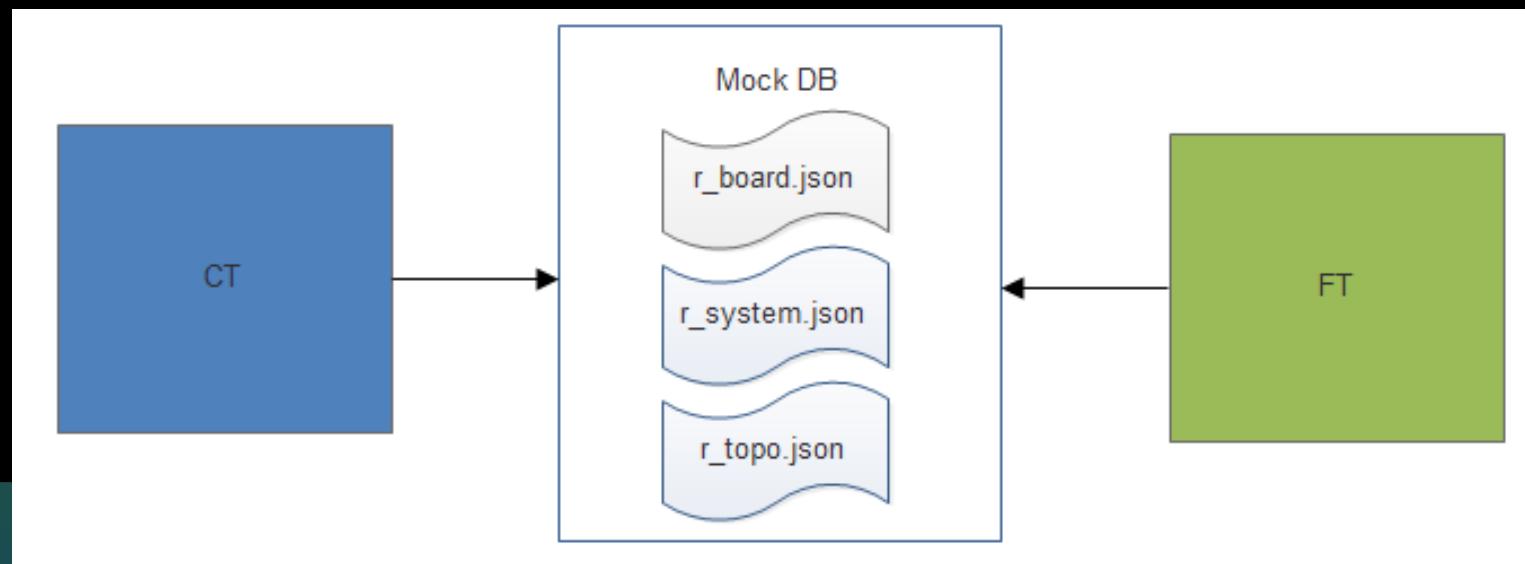
最佳实践三：内存数据库

挑战

- 内存数据库表之间关系复杂，普通开发人员很难维护
- 内存数据库升级成本大

解决方案

- 数据库对象通过简单工厂模式来构造，当 CT 模式时选择不同的变化方向
- 每张表都是独立的，用 json 来表达，普通开发人员易维护



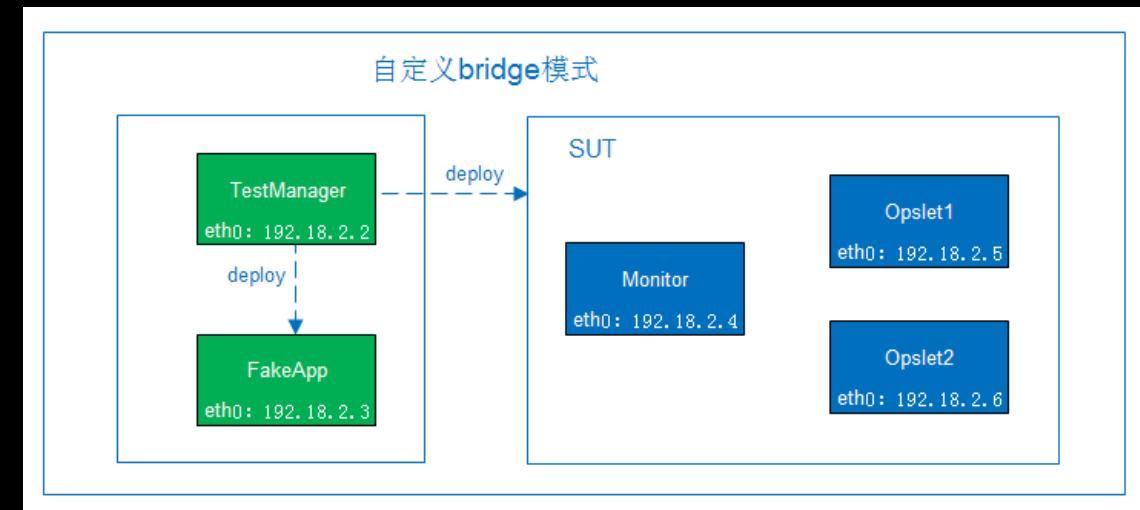
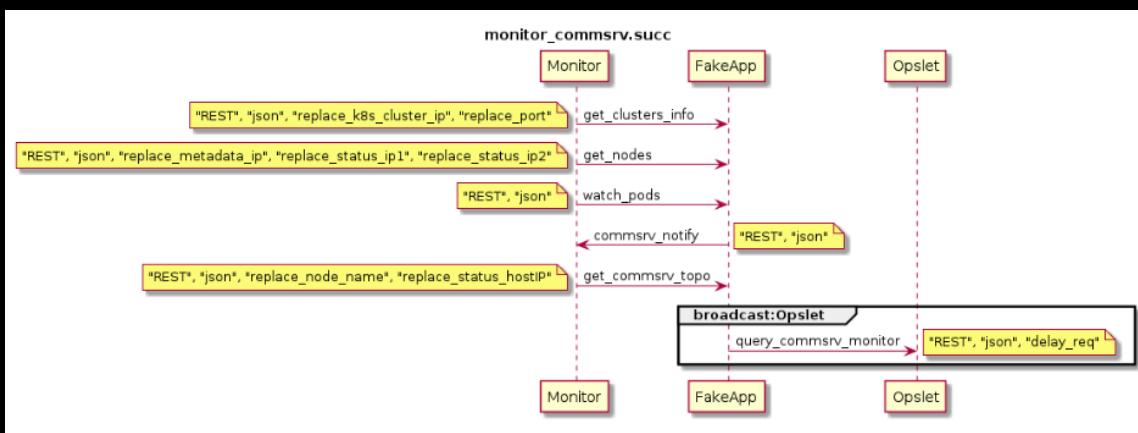
最佳实践四：SUT 分布式

挑战

- SUT 中部分服务单实例，部分服务多实例

解决方案

- 支持自定义 bridge 模式，为所有容器分配 IP



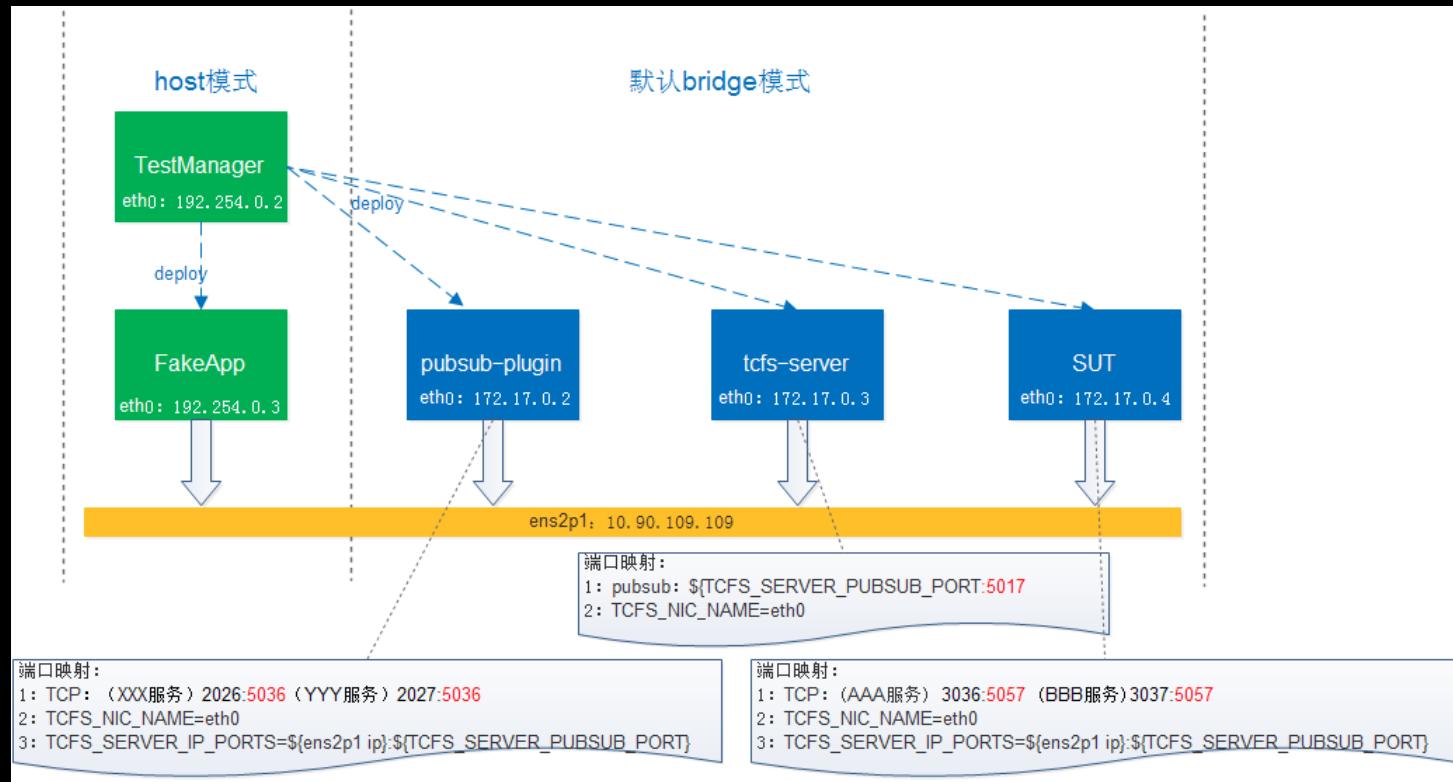
最佳实践五：并发

挑战

- 在一个单板内并发的跑多组 UT, 多组 FT, 多组 CT, 如何隔离?
- 电信微服务平台中某些端口是写死的, 比如协议栈的 TCP 端口和消息总线的 Serer 端口, 如何并发多组 CT?

解决方案

- CT 容器通过命名空间, 精准管理其生命周期, 不能误杀 UT 容器和 FT 容器, 也不能误杀其他组的 CT 容器
- 通过容器默认的 bridge 模式来隔离端口, Coral 不感知



最佳实践六：二进制码流

挑战

- 电信微服务总线发送消息的 payload 为 json 或二进制码流，二进制码流如何表达和编码？

```
{  
    "Msgbrief": {  
        "SessionType": "RUL_REROUTE_SESSION",  
        "MsgId": 1,  
        "SessionInst": "00000005",  
        "Version": 64,  
        "Length": 100,  
        "Priority": 2  
    },  
    "Payload": "[89 01 00 01 00 00 00 00 01 3d 10 2c ff]",  
    "parameters": []  
}
```

解决方案

- 自定义标准：通过以 "[" 开始以 "]" 结束的十六进制码流字符串来表达二进制码流
- FakeApp 发送二进制码流：
 - a. 去掉 "[" 和 "]"
 - b. 去掉字符串双引号
 - c. base64 编码
 - d. 序列化；
- FakeApp 接收二进制码流：
 - a. 反序列化
 - b. base64 解码
 - c. 转成十六进制码流字符串 (actual)
 - d. 取出该服务契约实例中的十六进制码流字符串 (expect) 进行断言

03

中兴契约测试工程管理实践

契约测试试点及推广，契约测试嵌入团队研发流程，契约测试成果和收益

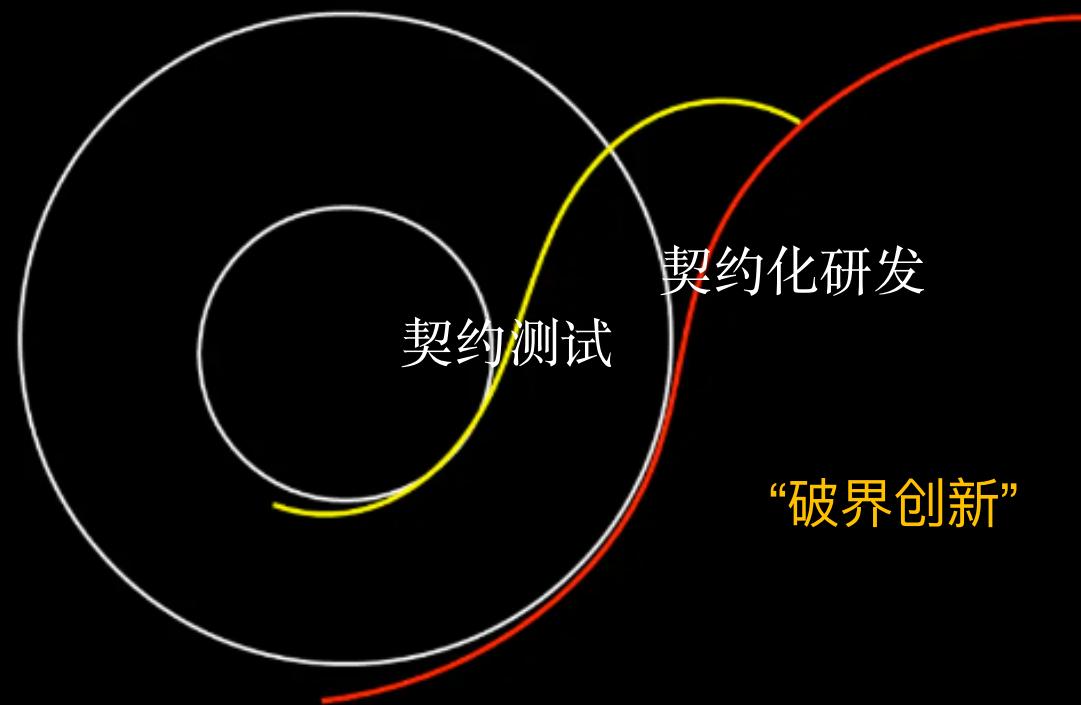
契约测试试点：自下而上

- 寻找游击队
- 确定试点团队
- 梳理典型用例
- 研发 Coral 原型
- 上线 CI

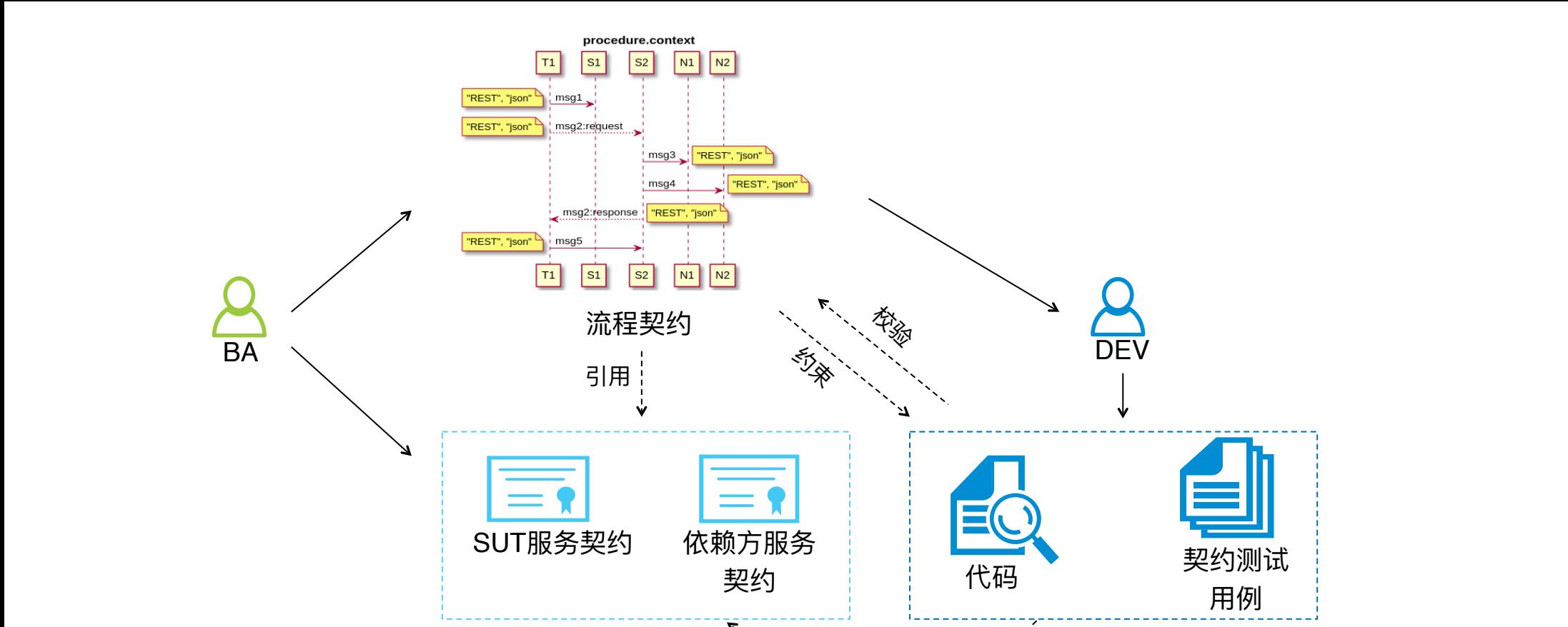
契约测试推广：自上而下

- 组建正规军
- 规划年度目标
- 多级任务管理
- 发布 Coral 版本
- 完善契约中心

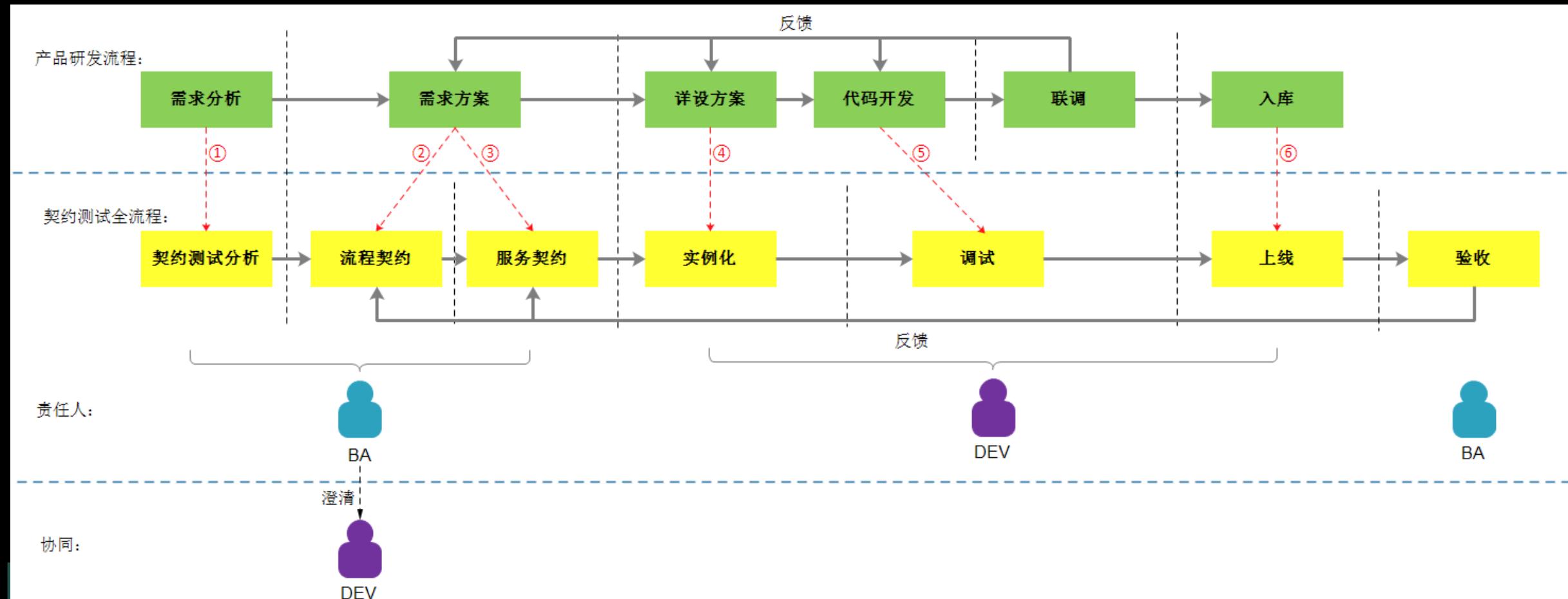
契约化研发



契约是 BA 和 DEV 的粘合剂



契约测试嵌入团队研发流程



成果

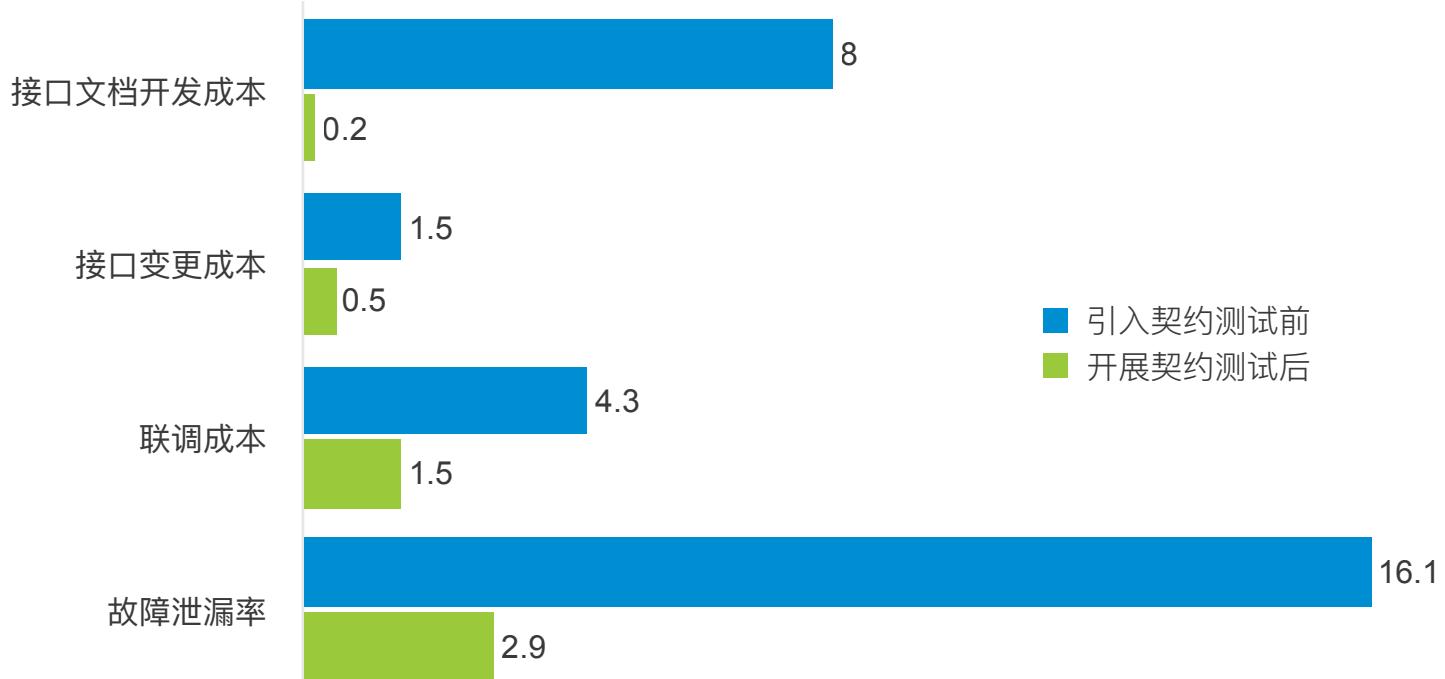
容器云平台项目

- 已上线服务 40 个
- 已完成用例 438 个，已上线用例 421 个

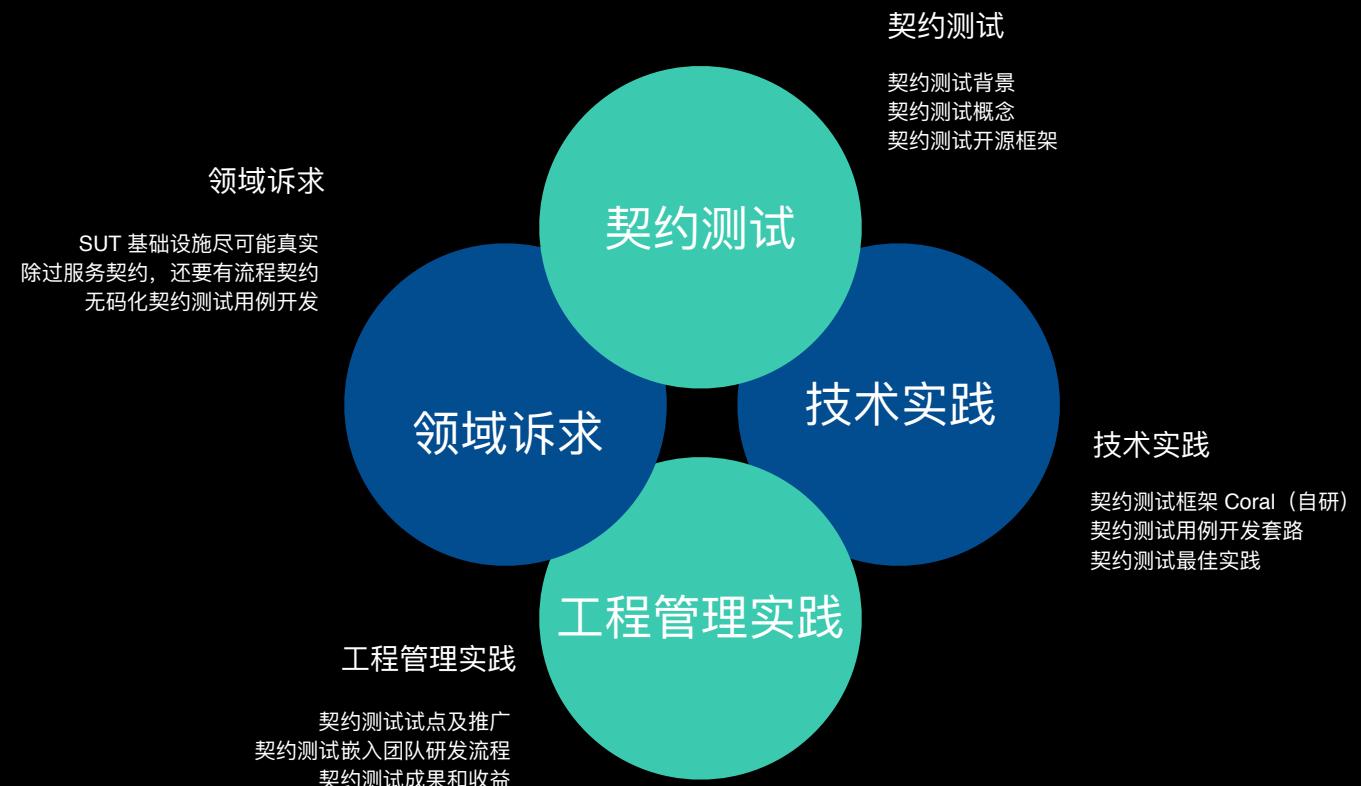
电信微服务平台项目

- 已上线服务 15 个
- 已完成用例 377 个，已上线用例 332 个

收益



小结



谢谢

张银奎

知名系统内核专家，《软件调试》作者



格蠹科技 (Xedge.ai) 创始人，专注AI边缘计算。资深软硬件架构师，拥有13年 Intel 工作经历，10年 GPU 深度优化技术积累。对机器学习、操作系统内核、驱动程序、虚拟化技术、软件调优有深入研究。毕业于上海交通大学，著有《软件调试》、《格蠹汇编》。

主办方：

Boolan
高端IT咨询与教育平台

C++ Summit 2020

张银奎

格蠹科技创始人

从纳秒级优化谈CPU 眼里的好代码

张银奎

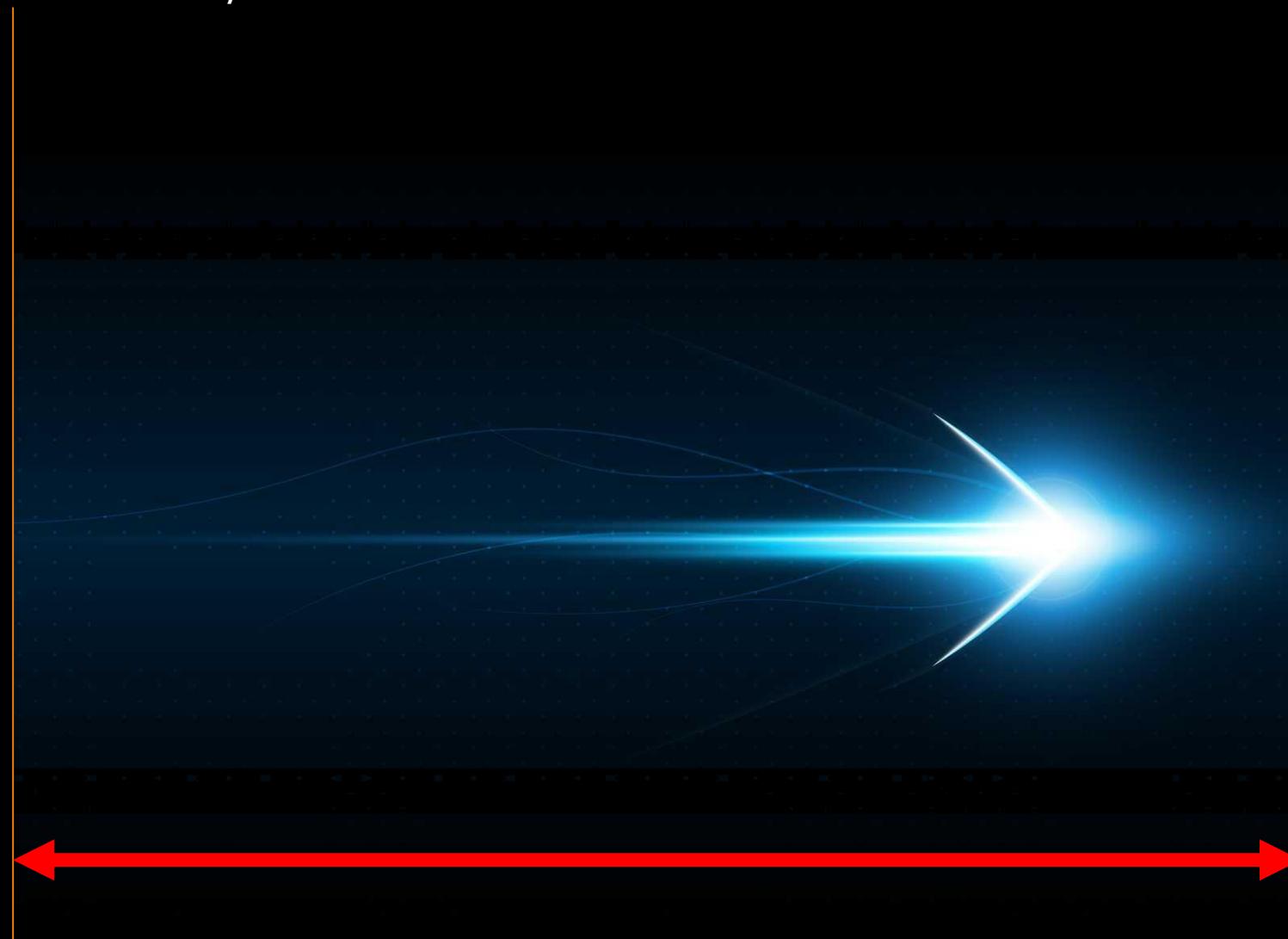
- Raymond Zhang
- 20年编程经历
 - 格蠹科技
 - 2003.5 ~ 2016.12 INTEL
- 2005年参加第一届C++大会
 - 《异常处理得与失》
- 《软件调试》作者
 - +《格蠹汇编》
- 译作
 - 《21世纪机器人》
 - 《机器学习》
 - 《数据挖掘原理》
 - 《人工智能——求解复杂问题的方法和策略》
 - 《观止——微软创建NT和未来的夺命狂奔》
 - 《现代x86汇编语言程序编程》



苏轼：“问汝平生功业，黄州惠州儋州。”

$c_0=299792.458\text{km/s}$

CPP-Summit 2020



1纳秒时间里光在真空中只可以行进大约30CM

问题1：

CPU执行下面两条语句的速度一样么？

`A = B * 0.01;`

`A = B / 100.0;`

```
float ge_mul_div(int count)
{
    int ret = 0;
    uint64_t start, end;
    float sum = 0;

    start = __rdtsc();
    for (int n = 0; n < count; n++)
        sum += (n*0.01);
    end = __rdtsc();

    cout << "sum of i*0.01 " << count << " took " << end - start << " ticks. " << sum << endl;

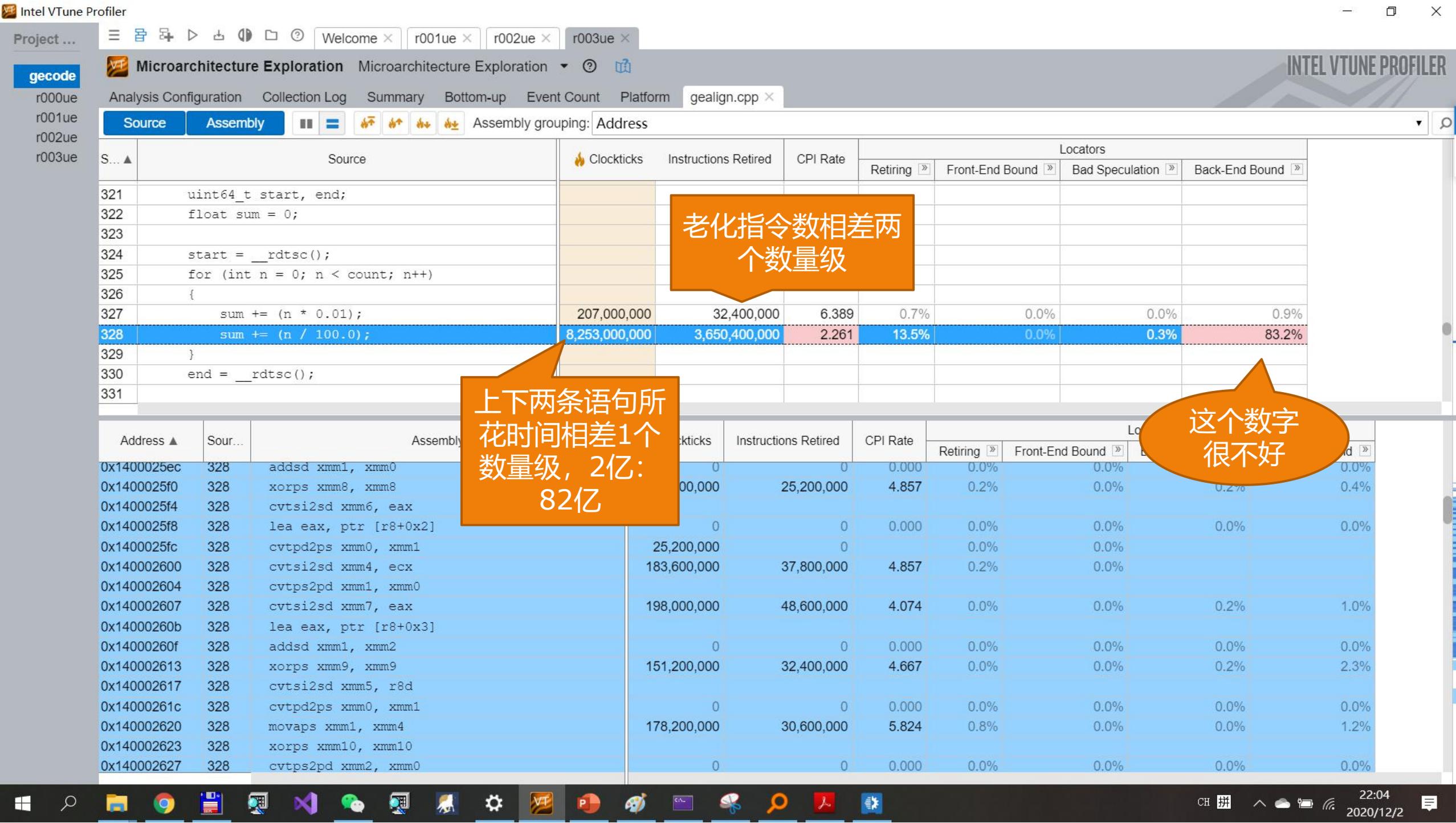
    sum = 0;
    start = __rdtsc();
    for (int n = 0; n < count; n++)
        sum += (n / 100.0);
    end = __rdtsc();

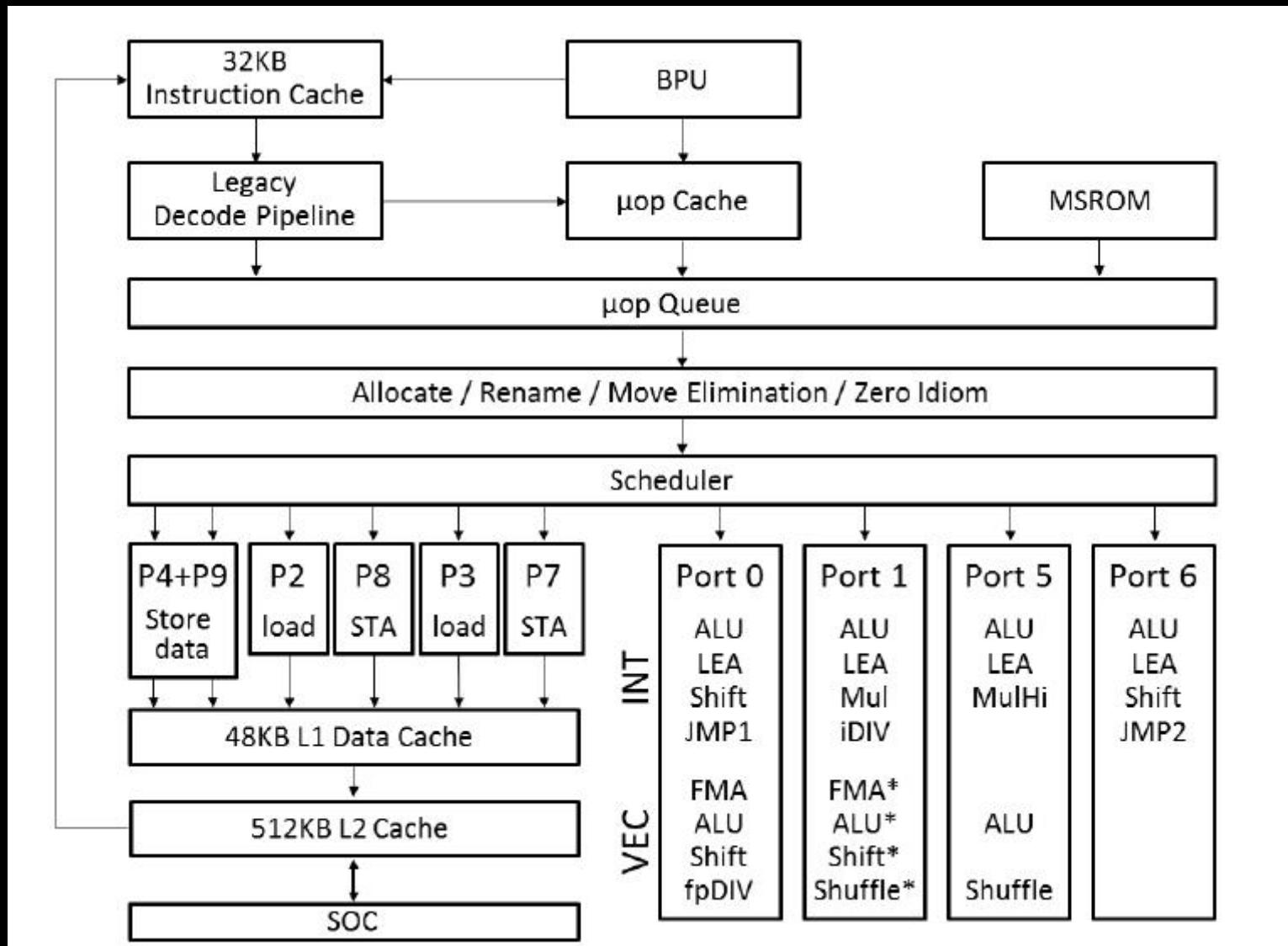
    cout << "sum of i/100.0 " << count << " took " << end - start << " ticks. " << sum << endl;
    return sum;
}
```

实验1

```
sum of i*0.01 1000000 took 15824442 ticks. 4.99761e+09
sum of i/100.0 1000000 took 23897285 ticks. 4.99761e+09
sum of i*0.01 1000000 took 17029383 ticks. 4.99761e+09
sum of i/100.0 1000000 took 24590475 ticks. 4.99761e+09
sum of i*0.01 1000000 took 16524510 ticks. 4.99761e+09
sum of i/100.0 1000000 took 23933803 ticks. 4.99761e+09
```

- 把循环变量i与0.01相乘1百万次用时大约8毫秒
- 把循环变量i除以100.0， 1百万次用时大约12毫秒
- 把 $i / 100.0$ 改为等价的 $i * 0.01$ 可以提高速度 $(12-8)/12 = 33\%$



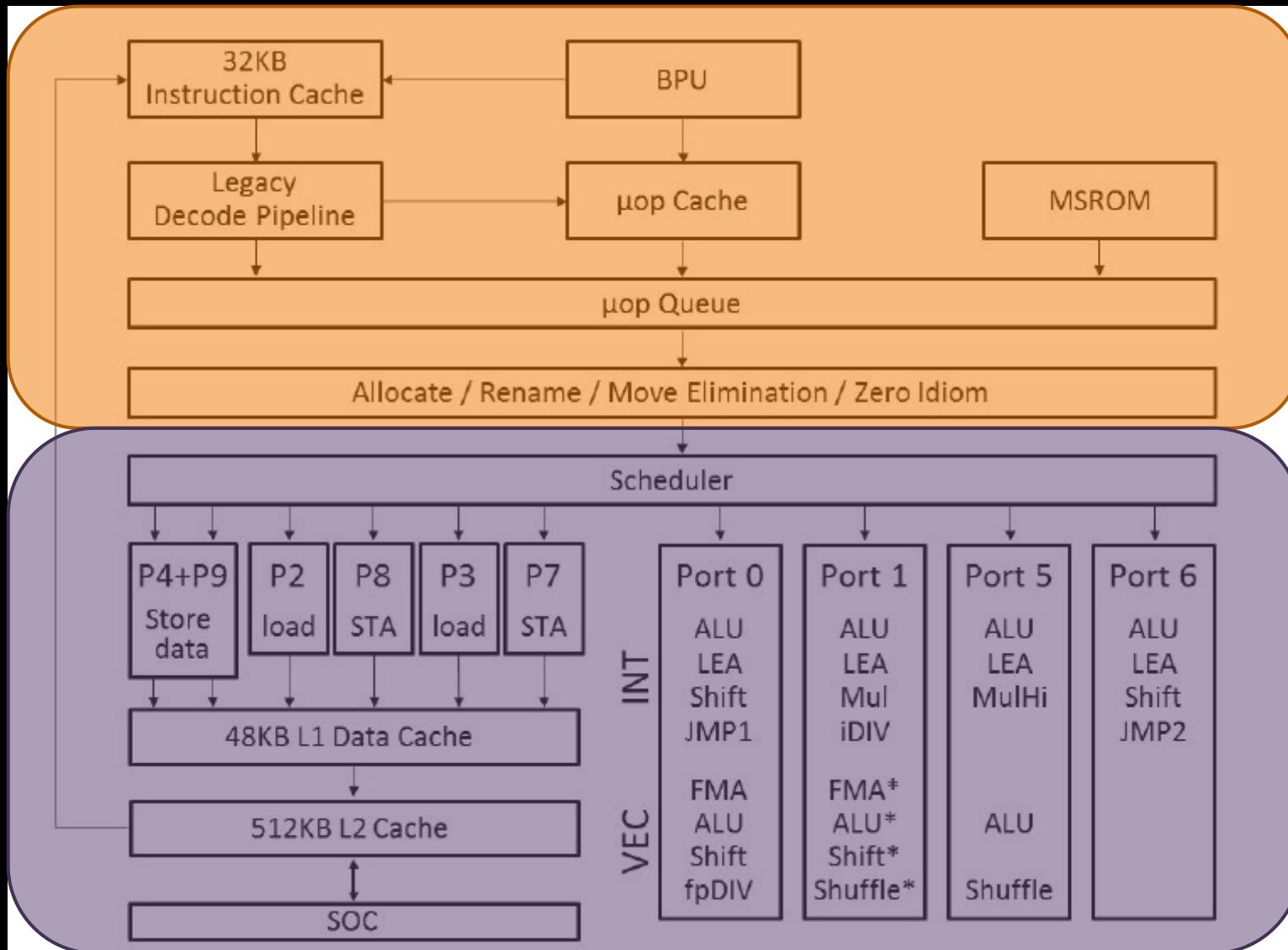


Ice Lake Client
Microarchitecture

前端

后端

乱序执行引擎



前端的主要部件

- Branch Predictor Unit (BPU)
- Loop Stream Detector (LSD)
- Decoded I-cache (DSB).
- Microcode sequencer (MS)
- MS – ROM

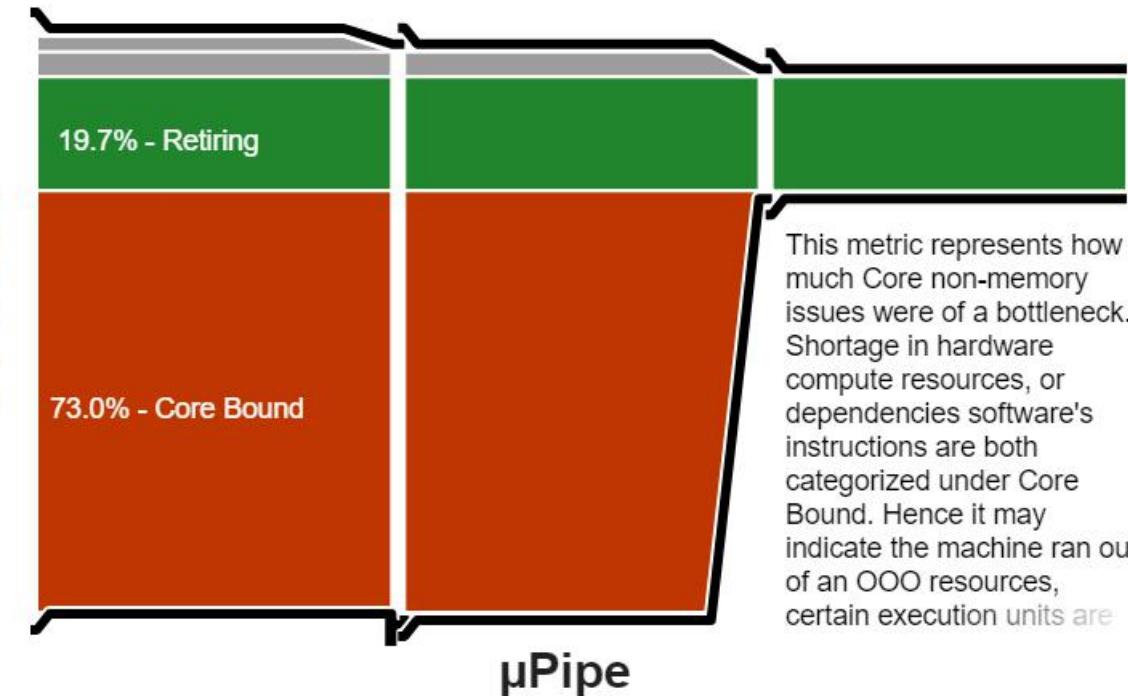


Microarchitecture Exploration Microarchitecture Exploration ▾ ⓘ

Analysis Configuration Collection Log Summary Bottom-up Event Count Platform

Elapsed Time[?]: 11.283s ⏱

| | |
|--|---------------------------|
| Clockticks: | 20,084,400,000 |
| Instructions Retired: | 11,712,600,000 |
| CPI Rate [?] : | 1.715 ↘ |
| MUX Reliability [?] : | 0.965 |
| Retiring [?] : | 19.7% of Pipeline Slots |
| Front-End Bound [?] : | 2.7% of Pipeline Slots |
| Bad Speculation [?] : | 0.3% of Pipeline Slots |
| Back-End Bound [?] : | 77.4% ↗ of Pipeline Slots |
| Memory Bound [?] : | 4.4% of Pipeline Slots |
| Core Bound [?] : | 73.0% ↗ of Pipeline Slots |
| Divider [?] : | 22.5% ↗ of Clockticks |
| Port Utilization [?] : | 82.4% ↗ of Clockticks |
| Cycles of 0 Ports Utilized [?] : | 23.4% ↗ of Clockticks |
| Cycles of 1 Port Utilized [?] : | 19.5% of Clockticks |
| Cycles of 2 Ports Utilized [?] : | 4.4% of Clockticks |
| Cycles of 3+ Ports Utilized [?] : | 2.6% of Clockticks |
| Vector Capacity Usage (FPU) [?] : | 25.0% ↗ |
| Average CPU Frequency [?] : | 1.8 GHz |
| Total Thread Count: | 4 |
| Paused Time [?] : | 0s |



This metric represents how much Core non-memory issues were of a bottleneck. Shortage in hardware compute resources, or dependencies software's instructions are both categorized under Core Bound. Hence it may indicate the machine ran out of an OOO resources, certain execution units are

μPipe

This diagram represents inefficiencies in CPU usage. Treat it as a pipe with an output flow equal to the "pipe efficiency" ratio: (Actual Instructions Retired)/(Maximum Possible Instruction Retired). If there are pipeline stalls decreasing the pipe efficiency, the pipe shape gets more narrow.

Core Bound

Front-end Bound

Store Bound L1 Bound

L2 Bound Back-end Bound

L3 Bound Memory Bound

DRAW Bound

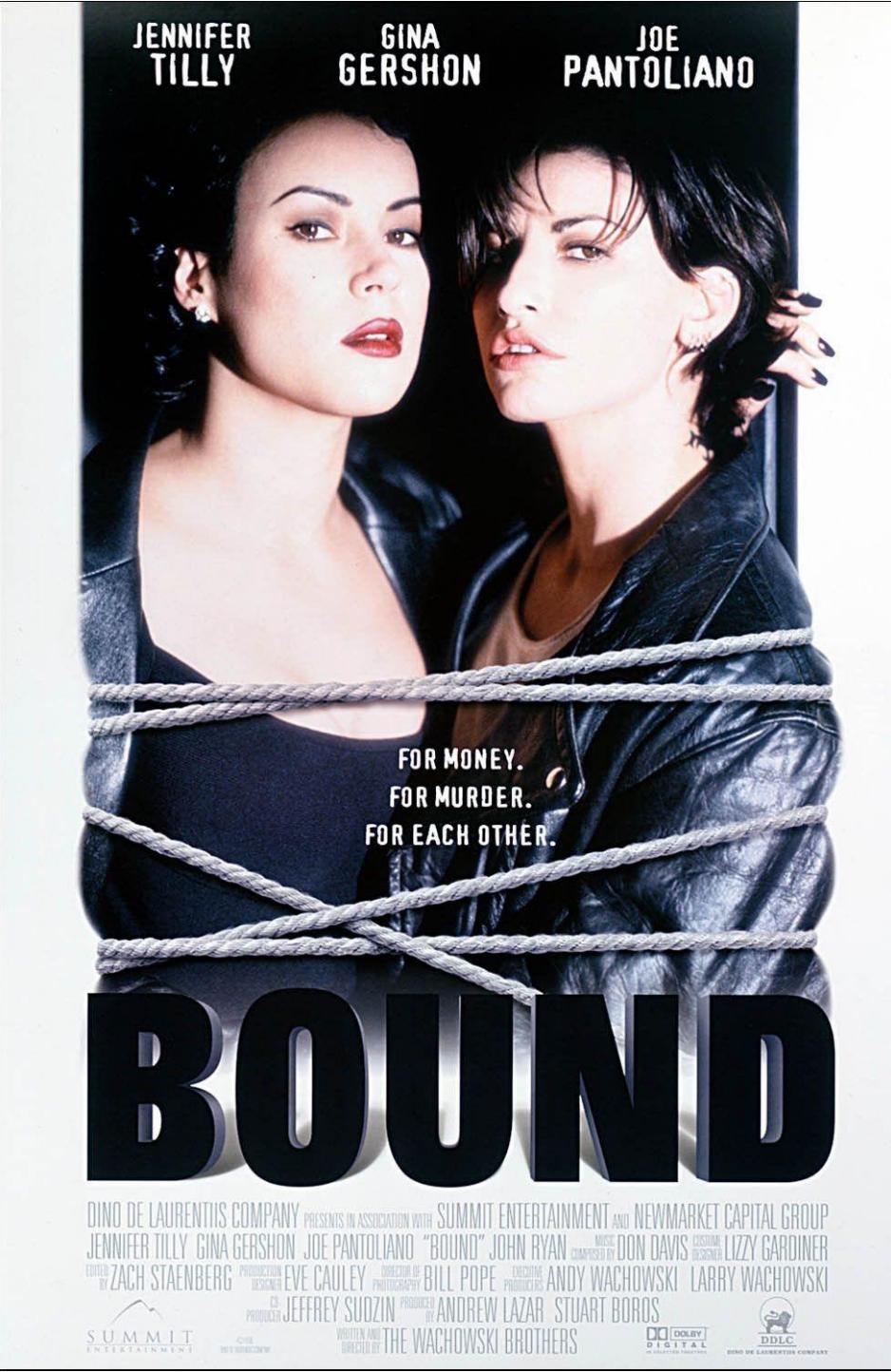
A decorative bar chart consisting of vertical bars of varying heights, colored in a gradient from dark teal to light green, located on the right side of the slide.

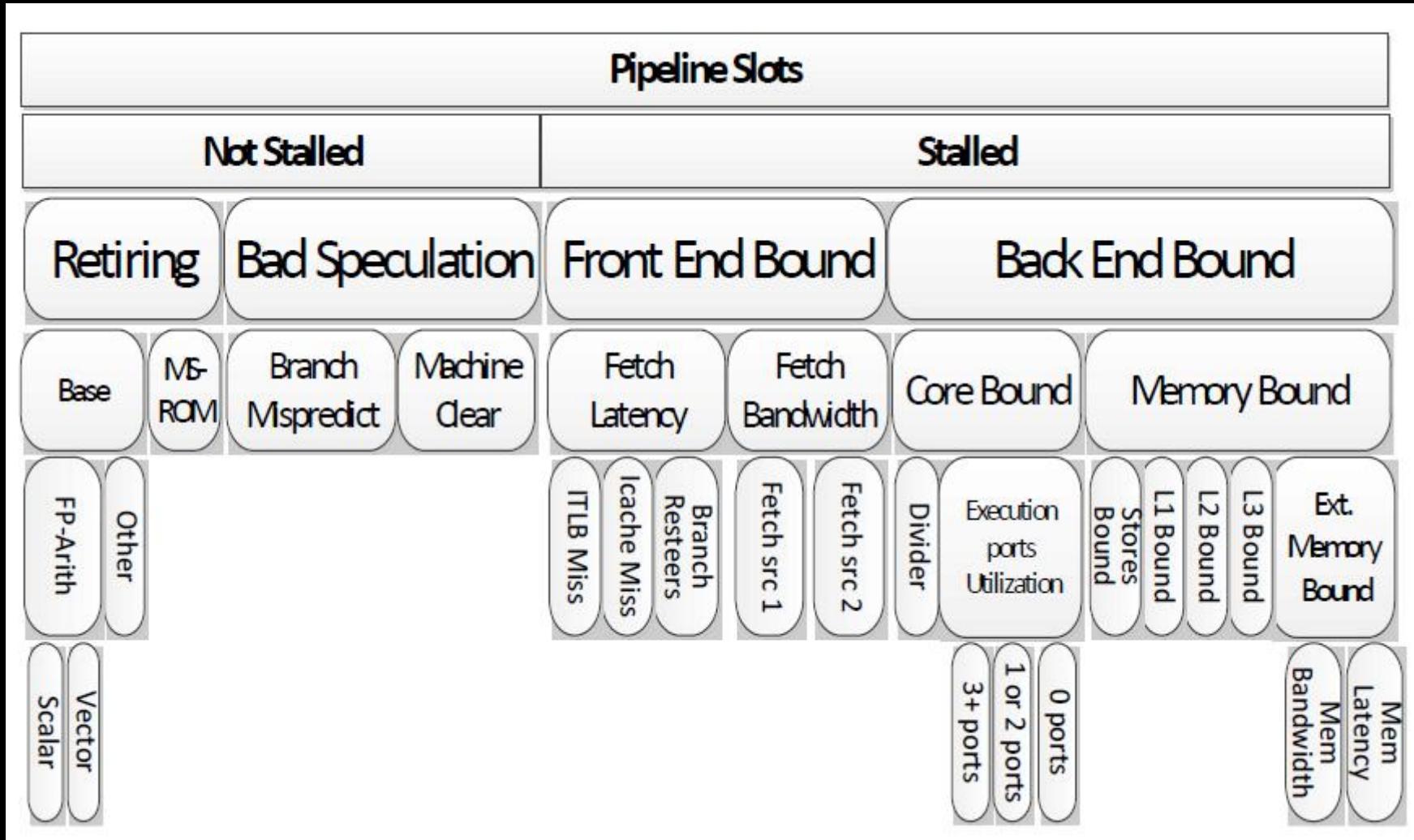
BOUND

性能优化中的常用词

绑定

约束

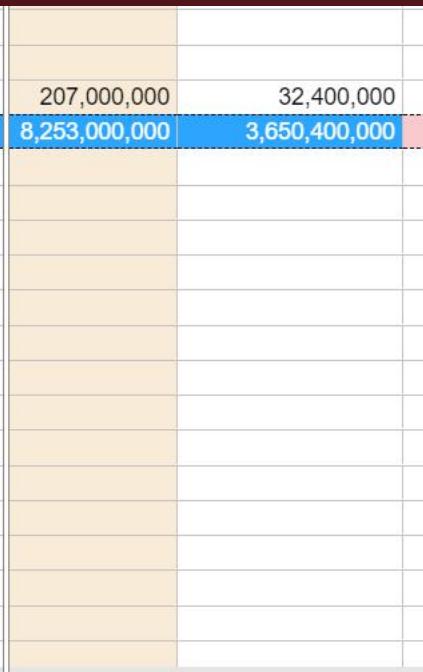




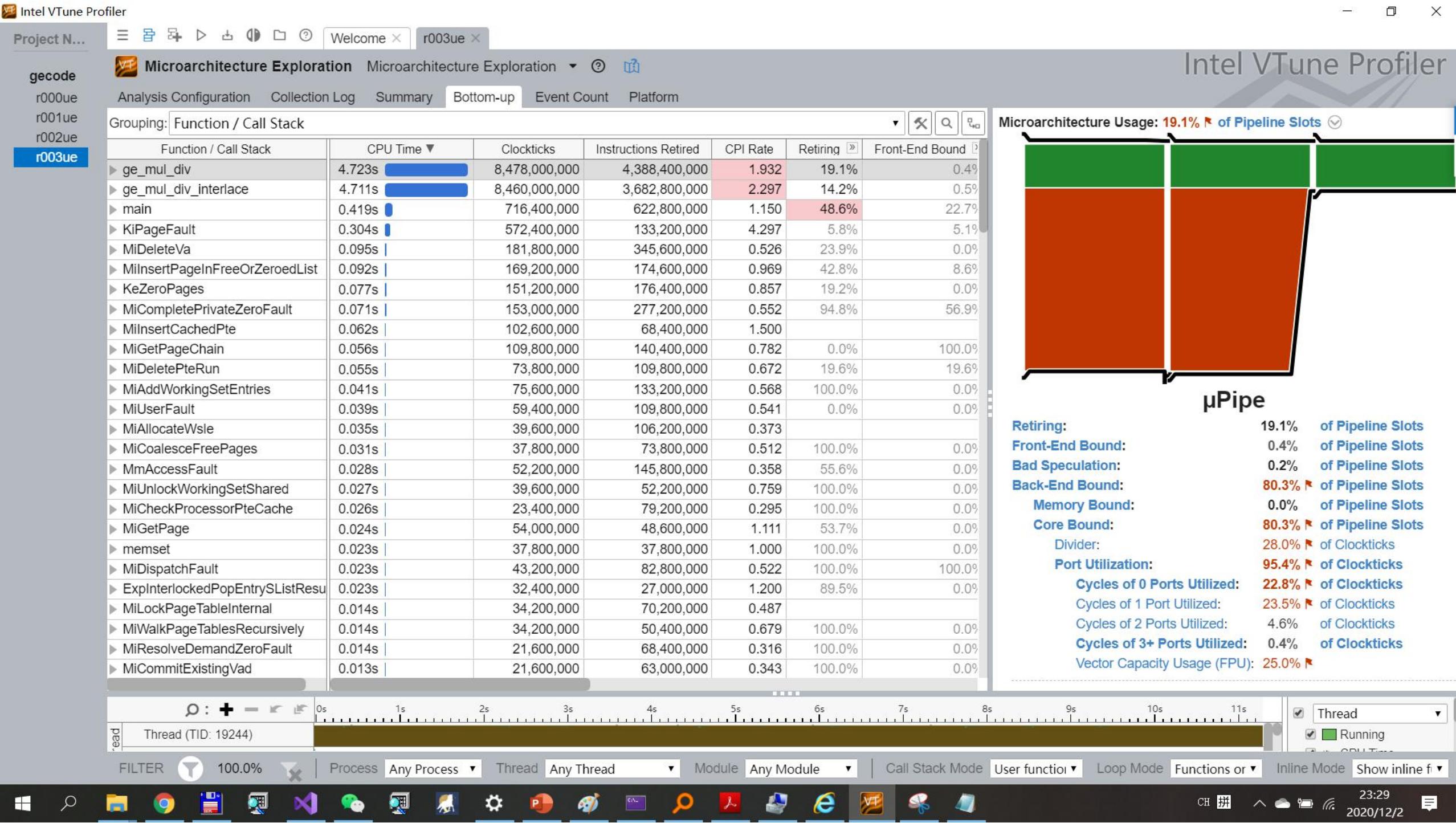
A significant portion of pipeline slots are remaining empty. When operations take too long in the back-end, they introduce bubbles in the pipeline that ultimately cause fewer pipeline slots containing useful work to be retired per cycle than the machine is capable to support. This **opportunity cost** results in slower execution. Long-latency operations like divides and memory operations can cause this, as can **too many operations being directed to a single execution port** (for example, more multiply operations arriving in the back-end per cycle than the execution unit can support).

```
325     for (int n = 0; n < count; n++)  
326     {  
327         sum += (n * 0.01);  
328         sum += (n / 100.0);  
329     }  
330     end = __rdtsc();  
331  
332  
333     cout << "mul div interlace " << count << " times"  
334     return sum;  
335 }
```

```
336  
337 // most stupid method to determine prime no  
338 bool is_prime(uint64_t n)  
339 {  
340     uint64_t start, end, qroot;  
341     uint64_t remains, counter = 0;  
342  
343     qroot = sqrt(n);
```



A significant portion of pipeline slots are remaining empty. When operations take too long in the back-end, they introduce bubbles in the pipeline that ultimately cause fewer pipeline slots containing useful work to be retired per cycle than the machine is capable to support. This opportunity cost results in slower execution. Long-latency operations like divides and memory operations can cause this, as can too many operations being directed to a single execution port (for example, more multiply operations arriving in the back-end per cycle than the execution unit can support).



端口的能力是不一样的

| Port 0 | Port 1 ¹ | Port 2 | Port 3 | Port 4 | Port 5 | Port 6 | Port 7 | Port 8 | Port 9 |
|---------------------------------------|--|--------|--------|------------|---------------------------------|--------------------------------------|---------------|---------------|------------|
| INT ALU LEA INT Shift Jump1 | INT ALU LEA INT Mul INT Div | Load | Load | Store Data | INT ALU LEA INT MUL Hi | INT ALU LEA INT Shift Jump2 | Store Address | Store Address | Store Data |
| FMA Vec ALU Vec Shift FP Div | FMA* Vec ALU* Vec Shift* Vec Shuffle* | | | | Vec ALU Vec Shuffle | | | | |

Dispatch Port and Execution Stacks of the Ice Lake Client Microarchitecture

| Execution Unit | # of Unit | Instructions |
|----------------|-----------|---|
| ALU | 4 | add, and, cmp, or, test, xor, movzx, movsx, mov, (v)movdqu, (v)movdqqa, (v)movap*, (v)movup* |
| SHFT | 2 | sal, shl, rol, adc, sarx, adcx, adox, etc. |
| Slow Int | 1 | mul, imul, bsr, rcl, shld, mulx, pdep, etc. |
| BM | 2 | andn, bextr, blsi, blsmask, bzhi, etc. |
| Vec ALU | 3 | (v)pand, (v)por, (v)pxor, (v)movq, (v)movq, (v)movap*, (v)movup*, (v)andp*, (v)orp*, (v)paddb/w/d/q, (v)blendv*, (v)blendp*, (v)pblendd |
| Vec_Shft | 2 | (v)psllv*, (v)psrlv*, vector shift count in imm8 |
| Vec Add | 2 | (v)addp*, (v)cmpp*, (v)max*, (v)min*, (v)padds*, (v)paddus*, (v)psign, (v)pabs, (v)pavgb, (v)pcmpeq*, (v)pmax, (v)cvt�2dq, (v)cvtdq2ps, (v)cvtsd2si, (v)cvtss2si |
| Shuffle | 2 | (v)shufp*, vperm*, (v)pack*, (v)unpck*, (v)punpck*, (v)pshuf*, (v)pslldq, (v)alignr, (v)pmovzx*, vbroadcast*, (v)pslldq, (v)psrldq, (v)pblendw |
| Vec Mul | 2 | (v)mul*, (v)pmul*, (v)pmadd* |
| SIMD Misc | 1 | STTNI, (v)pclmulqdq, (v)psadw, vector shift count in xmm |
| FP Mov | 1 | (v)movsd/ss, (v)movd gpr |
| DIVIDE | 1 | divp*, divs*, vdiv*, sqrt*, vsqrt*, rcp*, vrcp*, rsqrt*, idiv |

只有一个执行单元可以做除法

Ice Lake Client Microarchitecture Execution Units and Representative Instructions

对比

| Instruction | Throughput | | Latency | |
|-----------------------------|------------------------|--|------------------------|--|
| | 06_5CH, 5FH, 7AH | 06_37H, 4AH,4CH ,4DH,5A H,5DH | 06_5CH, 5FH, 7AH | 06_37H, 4AH,4CH, 4DH,5AH, 5DH |
| IDIV r32 | 12-25 | 35-47 | 12-25 | 35-47 |
| IDIV r64 | 12-41 | 49-135 | 12-41 | 49-135 |
| IMUL r32, r32 (single dest) | 1 | 1 | 3 | 3 |
| IMUL r32 (dual dest) | 2 | 5 | 3 (4, EDX) | 4 |
| IMUL r64, r64 (single dest) | 2 | 2 | 5 | 5 |
| IMUL r64 (dual dest) | 2 | 4 | 5 (6,RDX) | 5 (7,RDX) |

执行除法的流水线很长，
需要的时钟周期是乘法的4-40倍

Instructions Latency and Throughput Recent Microarchitectures for Intel Atom Processors

```
float ge_mul_div_interlace_para(int count)
{
    int ret = 0;
    uint64_t start, end;
    float sum1 = 0, sum2 =0;

    start = __rdtsc();
    for (int n = 0; n < count; n++)
    {
        sum1 += (n * 0.01);
        sum2 += (n / 100.0);
    }
    end = __rdtsc();

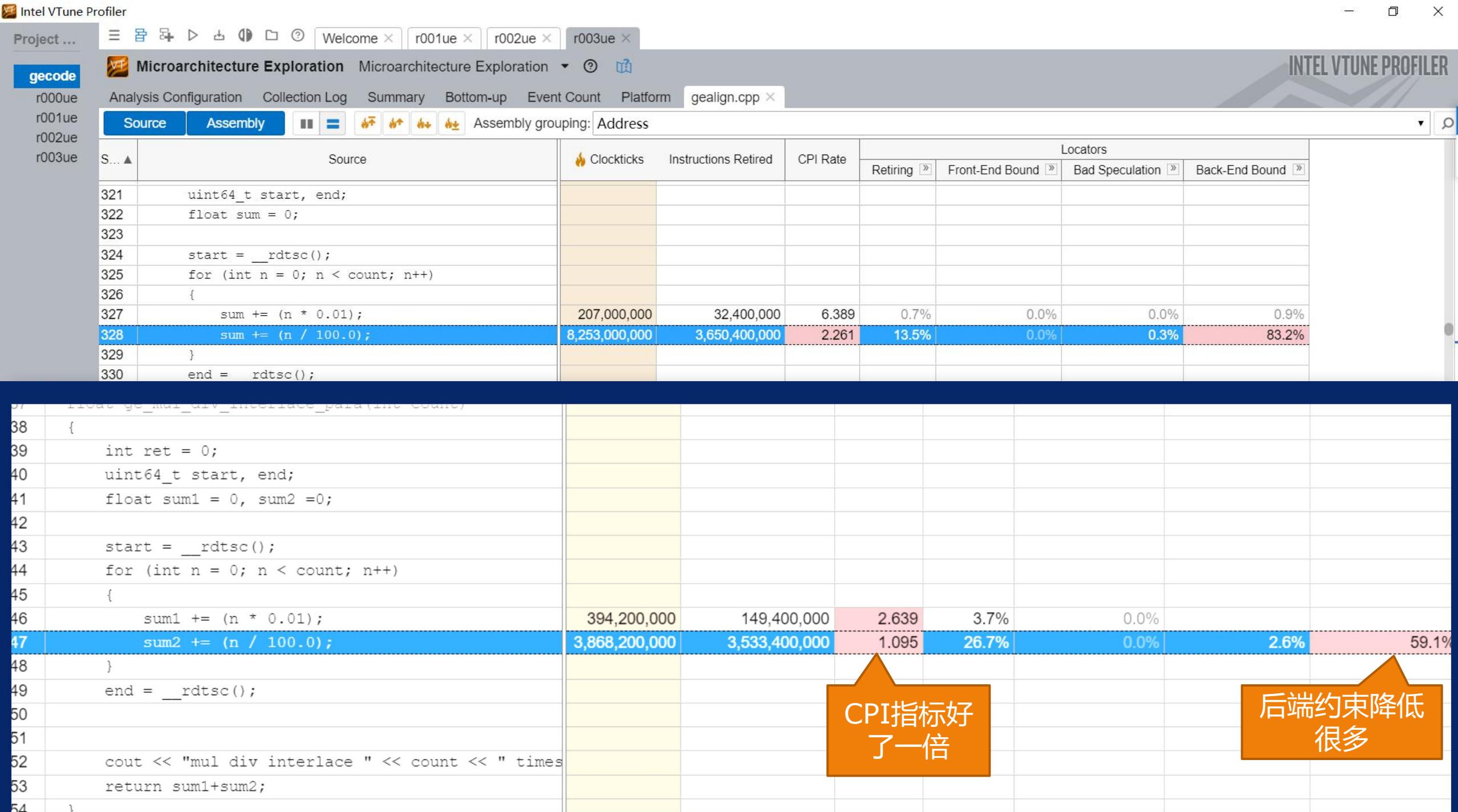
    cout << "parallel interlace " << count << " times took "
        << end - start << " ticks. " << sum1 + sum2 << endl;
    return sum1+sum2;
}
```

这样改代码会好一些吗?

```
sum of i*0.01 1000000 took 15793825 ticks. 4.99761e+09
sum of i/100.0 1000000 took 24295920 ticks. 4.99761e+09
mul div interlace 1000000 times took 32069874 ticks. 9.99448e+09
parallel interlace 1000000 times took 28989843 ticks. 9.99522e+09
sum of i*0.01 1000000 took 16498349 ticks. 4.99761e+09
sum of i/100.0 1000000 took 23859965 ticks. 4.99761e+09
mul div interlace 1000000 times took 34116224 ticks. 9.99448e+09
parallel interlace 1000000 times took 29325904 ticks. 9.99522e+09
sum of i*0.01 1000000 took 16209737 ticks. 4.99761e+09
sum of i/100.0 1000000 took 23928995 ticks. 4.99761e+09
mul div interlace 1000000 times took 31938441 ticks. 9.99448e+09
parallel interlace 1000000 times took 28976150 ticks. 9.99522e+09
```

```
0:000> ?? 100+(31938441-28976150)/31938441.0*100
double 109.27500187000360654
```

大约提高了9.2%





C君直言1：

我天生不喜欢除法，做起来别扭，
费劲，非让我做，就别嫌慢啊

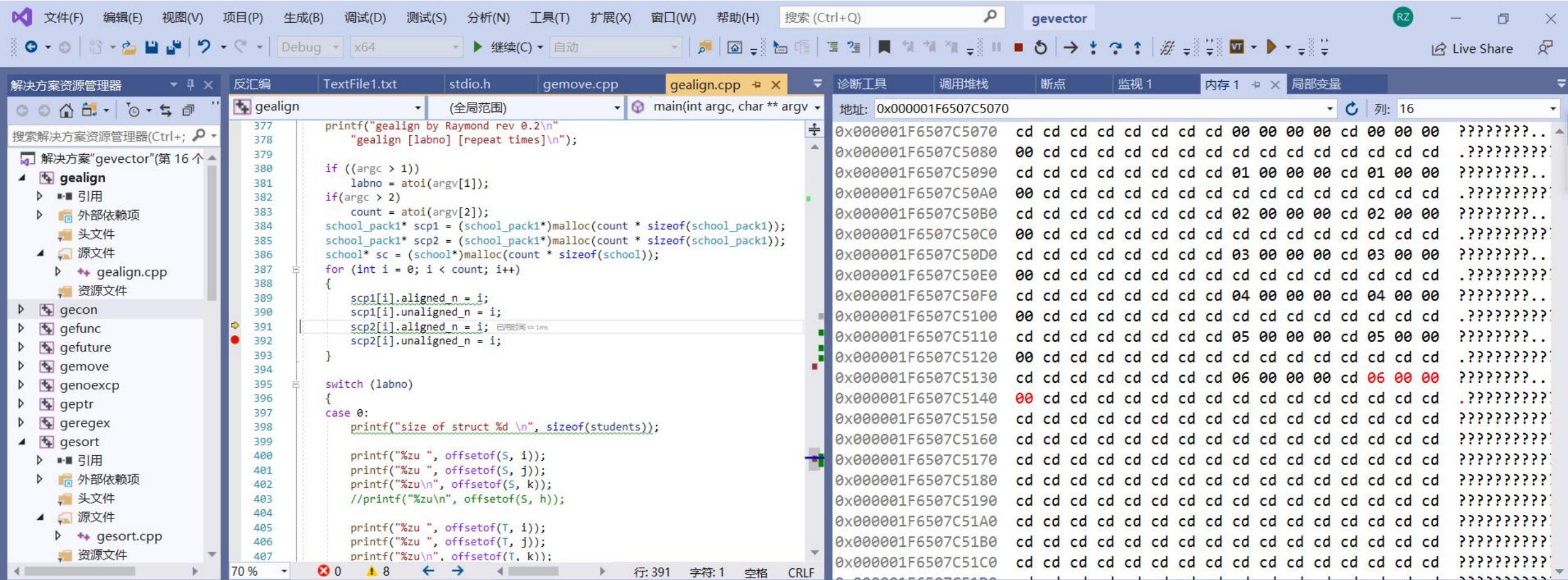
问题2：

写右侧school_s结构体的i和n成员的速度一样么？

school.i = x;

school.n = x;

```
#pragma pack (push)
#pragma pack (1)
struct school_s
{
    double a;
    int i;
    char h;
    int n;
    uint64_t c;
    char d[7];
};
```



Intel VTune Profiler

Project ... Welcome x r006ue x

Microarchitecture Exploration Microarchitecture Exploration

Analysis Configuration Collection Log Summary Bottom-up Event Count Platform gealig

Source Assembly

Assembly grouping: Address

0:000> ?? (4986-3024)/4986.0
double 0.39350180505415160992

| S... L... | Source | Clockticks | Instructions Retired | CPI Rate | Locators | | | |
|--------------|---|-------------|----------------------|----------|----------|-----------------|-----------------|----------------|
| | | | | | Retiring | Front-End Bound | Bad Speculation | Back-End Bound |
| r000ue | | | | | | | | |
| r001ue | | | | | | | | |
| r002ue | | | | | | | | |
| r003ue | | | | | | | | |
| r004ue | | | | | | | | |
| r005ue | | | | | | | | |
| r006ue | | | | | | | | |
| 383 | count = atoi(argv[2]); | 302,400,000 | 131,400,000 | 2.301 | 5.8% | 5.8% | | |
| 384 | school_pack1* scp1 = (school_pack1*)malloc(count * sizeof(school)); | 498,600,000 | 262,800,000 | 1.897 | 3.8% | 0.0% | 1.0% | 27.2% |
| 385 | school_pack1* scp2 = (school_pack1*)malloc(count * sizeof(school)); | 342,000,000 | 153,000,000 | 2.235 | 1.9% | 0.0% | 6.7% | 11.1% |
| 386 | school* sc = (school*)malloc(count * sizeof(school)); | 469,800,000 | 219,600,000 | 2.139 | 4.8% | 0.0% | 0.0% | 14.6% |
| 387 | for (int i = 0; i < count; i++) | | | | | | | |
| 388 | { | | | | | | | |
| 389 | scp1[i].aligned_n = i; | | | | | | | |
| 390 | scp1[i].unaligned_n = i; | | | | | | | |
| 391 | scp2[i].aligned_n = i; | | | | | | | |
| 392 | scp2[i].unaligned_n = i; | | | | | | | |
| 393 | } | | | | | | | |

| Address ▲ | Sour... | Assembly | Clockticks | Instructions Retired | CPI Rate | Locators | | | |
|-------------|---------|-----------------------------------|-------------|----------------------|----------|----------|-----------------|-----------------|----------------|
| | | | | | | Retiring | Front-End Bound | Bad Speculation | Back-End Bound |
| 0x140002be0 | 387 | jle 0x140002c0a <Block 14> | | | | | | | |
| 0x140002be2 | | Block 12: | | | | | | | |
| 0x140002be2 | 387 | mov r8, r15 | | | | | | | |
| 0x140002be5 | 387 | lea rcx, ptr [r14+0xd] | | | | | | | |
| 0x140002be9 | 387 | sub r8, r14 | | | | | | | |
| 0x140002bec | 387 | nop dword ptr [rax], eax | | | | | | | |
| 0x140002bf0 | | Block 13: | | | | | | | |
| 0x140002bf0 | 389 | mov dword ptr [rcx-0x5], edx | 302,400,000 | 131,400,000 | 2.301 | 5.8% | 5.8% | | |
| 0x140002bf3 | 390 | mov dword ptr [rcx], edx | 498,600,000 | 262,800,000 | 1.897 | 3.8% | 0.0% | 1.0% | |
| 0x140002bf5 | 391 | mov dword ptr [r8+rcx*1-0x5], edx | 342,000,000 | 153,000,000 | 2.235 | 1.9% | 0.0% | 6.7% | |
| 0x140002bfa | 392 | mov dword ptr [r8+rcx*1], edx | 226,800,000 | 102,600,000 | 2.211 | 1.9% | 0.0% | 0.0% | |
| 0x140002bfe | 392 | lea rcx, ptr [rcx+0x1a] | 84,600,000 | 32,400,000 | 2.611 | 1.0% | 0.0% | 0.0% | |
| 0x140002c02 | 392 | inc edx | 77,400,000 | 32,400,000 | 2.389 | 1.0% | 0.0% | 0.0% | |
| 0x140002c04 | 392 | sub rbx, 0x1 | 81,000,000 | 52,200,000 | 1.552 | 1.0% | 0.0% | 0.0% | |
| 0x140002c08 | 392 | jnz 0x140002bf0 <Block 13> | 0 | 0 | 0.000 | 0.0% | 0.0% | 0.0% | |

写对齐的整数比不对齐的整数要快40%

gecode

r000ue

r001ue

r002ue

r003ue

r004ue

r005ue

r006ue

r007...

Elapsed Time [?]: 4.165s

Clockticks:

7,401,600,000

Instructions Retired:

7,158,600,000

CPI Rate [?]:

1.034 ↘

MUX Reliability [?]:

0.980

Retiring [?]:

25.5% ↘ of Pipeline Slots

General Retirement [?]:

19.6% of Pipeline Slots

Microcode Sequencer [?]:

5.9% ↘ of Pipeline Slots

Front-End Bound [?]:

9.8% of Pipeline Slots

Bad Speculation [?]:

2.4% of Pipeline Slots

Back-End Bound [?]:

62.4% ↘ of Pipeline Slots

Memory Bound [?]:

39.1% ↘ of Pipeline Slots

L1 Bound [?]:

12.5% ↘ of Clockticks

DTLB Overhead [?]:

2.4% of Clockticks

Loads Blocked by Store Forwarding [?]:

2.0% of Clockticks

Lock Latency [?]:

0.5% ↘ of Clockticks

Split Loads [?]:

0.0% of Clockticks

4K Aliasing [?]:

0.2% of Clockticks

FB Full [?]:

100.0% ↘ of Clockticks

L2 Bound [?]:

0.2% of Clockticks

L3 Bound [?]:

2.4% of Clockticks

DRAM Bound [?]:

6.9% of Clockticks

Store Bound [?]:

13.3% of Clockticks

Core Bound [?]:

23.3% ↘ of Pipeline Slots

Divider [?]:

0.0% of Clockticks

Port Utilization [?]:

21.0% ↘ of Clockticks

Cycles of 0 Ports Utilized [?]:

23.1% ↘ of Clockticks

Serializing Operations [?]:

21.2% ↘ of Clockticks

Cycles of 1 Port Utilized [?]:

7.4% ↘ of Clockticks

Cycles of 2 Ports Utilized [?]:

5.5% of Clockticks

Cycles of 3+ Ports Utilized [?]:

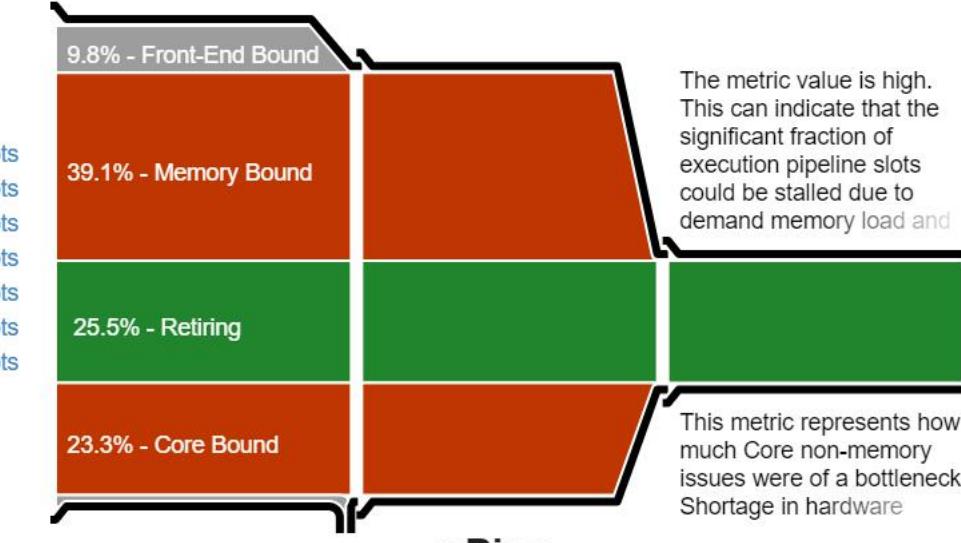
11.4% of Clockticks

Vector Capacity Usage (FPU) [?]:

0.0%

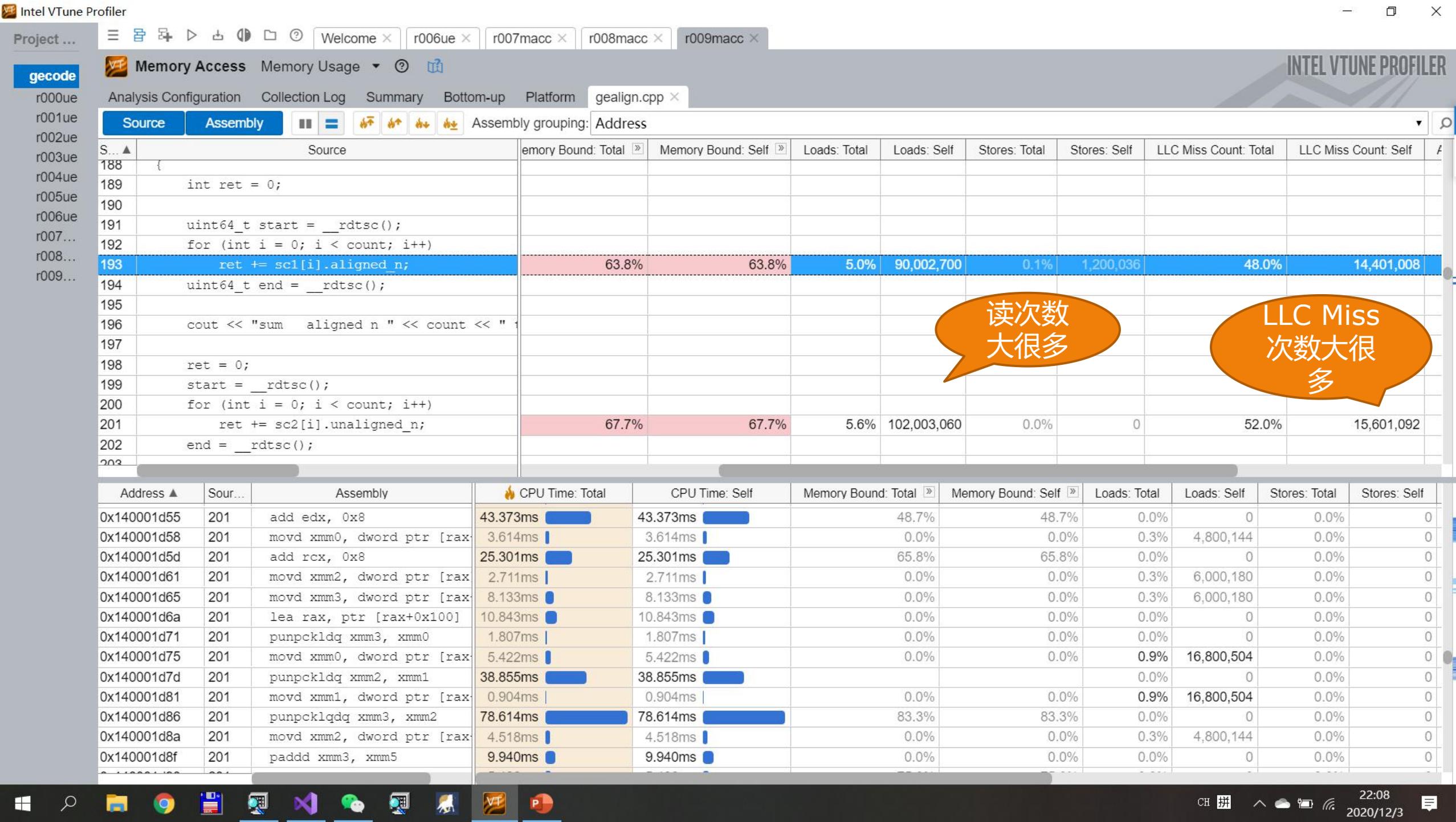
Average CPU Frequency [?]:

1.8 GHz



μPipe

This diagram represents inefficiencies in CPU usage. Treat it as a pipe with an output flow equal to the "pipe efficiency" ratio: (Actual Instructions Retired)/(Maximum Possible [Instruction Retired](#)). If there are pipeline stalls decreasing the pipe efficiency, the pipe shape gets more narrow.





C君直言2：

读写没有按边界对齐的整数会让我
做很多无用功，写比读更悬殊

问题3：

对于右侧这样的常量数据，下面哪种结构体定义好？

```
typedef struct _tagFOO_DATA_S {
    int nIndex_;
    char szName_[FOO_NAME_LENGTH];
} FOO_DATA_S;

typedef struct _tagGOO_DATA_S {
    int nIndex_;
    const char *szName_;
} GOO_DATA_S;
```

```
17  ↳FOO_DATA_S g_FooData[] = {
18  { 101 , " Euterpe oleracea "},
19  { 102 , " Malpighia emarginata "},
20  { 103 , " Irvingia gabonensis "},
21  { 104 , " Garcinia livingstonei "},
22  { 105 , " Elaeis guineensis "},
23  { 106 , " Cornus × unalaschkensis "},
24  { 107 , " Spondias dulcis "},
25  { 108 , " Elaeis oleifera "},
26  { 109 , " Prunus americana "},
27  { 110 , " Prunus armeniaca "},
28  { 111 , " Mangifera pajang "},
29  { 112 , " Prunus maritima "},
30  { 113 , " Antidesma bunius "},
31  { 114 , " Mangifera caesia "},
32  { 115 , " Prunus serotina "},
33  { 116 , " Parajubaea torallyi "},
34  { 117 , " Syzygium australe "},
35  { 118 , " Dacryodes edulis "},
36  { 119 , " Cornus canadensis "},
37  { 120 , " Casimiroa edulis "},
38  { 121 , " Eugenia reinwardtiana "},
39  { 122 , " Byrsonima crassifolia "},
40  { 123 , " Prunus avium "},
41  { 124 , " Elaeagnus multiflora "},
42  { 125 , " Eugenia involucrata "},
43  { 126 , " Ziziphus mauritiana "},
```

```
FOO_DATA_S* GetFooByID(FOO_DATA_S* foos, int nUnits, int id)
{
    FOO_DATA_S* ret = NULL;
    for (int i = 0; i < nUnits; i++)
        if (foos[i].nIndex_ == id)
            return &foos[i];

    return ret;
}

GOO_DATA_S* GetGooByID(GOO_DATA_S* goos, int nUnits, int id)
{
    GOO_DATA_S* ret = NULL;
    for (int i = 0; i < nUnits; i++)
        if (goos[i].nIndex_ == id)
            return &goos[i];

    return ret;
}
```

```
506 int GeLabIntactBatch(int nLoops)
507 {
508     printf("Cache lab to compare struct with names\n");
509     int interesed_ids[] = { 121,101,188,2,5,88,66,155,133,212,512,71,921,122,325,112,122,235 };
510     long long unsigned int start, end;
511     long long unsigned int sum_foo = 0, sum_goo = 0;
512     long extern_ref = 0;
513     GOO_DATA_S* goo;
514     FOO_DATA_S* foo;
515     start = __rdtsc();
516     for (int l = 0; l < nLoops; l++)
517     {
518         for (int i = 0; i < sizeof(interesed_ids) / sizeof(interesed_ids[0]); i++)
519         {
520             foo = GetFooByID(g_FooData, sizeof(g_FooData) / sizeof(g_FooData[0]), interesed_ids[i]);
521             if (foo)
522                 extern_ref += foo->szName_[0];
523         }
524     }
525     end = __rdtsc();
526     sum_foo += end - start;
527     start = __rdtsc();
528     for (int l = 0; l < nLoops; l++)
529     {
530         for (int i = 0; i < sizeof(interesed_ids) / sizeof(interesed_ids[0]); i++)
531         {
532             goo = GetGooByID(g_GooData, sizeof(g_GooData) / sizeof(g_GooData[0]), interesed_ids[i]);
533             if (goo)
534                 extern_ref += goo->szName_[0];
535         }
536     }
537     end = __rdtsc();
538     sum_goo += end - start;
539     printf("GeLabIntactBatch: %d loops is done\n"
540           "\tNonintact struct total %llu ticks aver %llu ticks\n"
541           "\tIntact    struct total %llu ticks aver %llu ticks\n",
542           nLoops, sum_foo, sum_foo / nLoops, sum_goo, sum_goo / nLoops
543       );
544     return extern_ref; // to avoid code cutting by compiler
545   }
```

```
GeCache [labno] [loops] rev6 by Raymond.  
Cache lab to compare struct with names  
GeLabIntactBatch: 1000000 loops is done  
    Nonintact struct total 4832459178 ticks aver 4832 ticks  
    Intact     struct total 3069835620 ticks aver 3069 ticks  
done with ret = 180000000
```

0:000> ?? (4832-3069)/4832.0
double 0.36485927152317881861

紧凑结构体的速度要比松散结构体的快36%

```
GeCache [labno] [loops] rev6 by Raymond.  
Cache lab to compare struct with names  
GeLabIntactInterlace: 1000000 loops is done  
    Nonintact struct total 5097826592 ticks aver 5097 ticks  
    Intact     struct total 3360932770 ticks aver 3360 ticks  
done with ret = 180000000
```

```
0:000> ?? (5097-3360)/5097.0  
double 0.34078869923484400584
```



| S... | Source | CPU Time: Total | CPU Time: Self | Memory Bound: Self | Loads: Total | Loads: Self | Stores: Total | Stores: Self |
|------|---|-----------------|----------------|--------------------|--------------|-------------|---------------|--------------|
| 509 | int interested_ids[] = { 121,101,188,2,5,88,66,15 | | | | | | | |
| 510 | long long unsigned int start, end; | | | | | | | |
| 511 | long long unsigned int sum_foo = 0, sum_goo = 0; | | | | | | | |
| 512 | long extern_ref = 0; | | | | | | | |
| 513 | GOO_DATA_S* goo; | | | | | | | |
| 514 | FOO_DATA_S* foo; | | | | | | | |
| 515 | start = __rdtsc(); | | | | | | | |
| 516 | for (int l = 0; l < nLoops; l++) | | | | | | | |
| 517 | { | | | | | | | |
| 518 | for (int i = 0; i < sizeof(interested_ids) / | 0.002s | 0.002s | 0.0% | 0.0% | 0 | 0.0% | 0 |
| 519 | { | | | | | | | |
| 520 | foo = GetFooByID(g_FooData, sizeof(g_Foo | 1.940s | 1.940s | 1.5% | 1.5% | 50.2% | 2,130,063... | 0.0% |
| 521 | if (foo) | 0.019s | 0.019s | | | 0.5% | 21,600,648 | 0.0% |
| 522 | extern_ref += foo->szName_[0]; | | | | | | | |
| 523 | } | | | | | | | |
| 524 | } | | | | | | | |
| 525 | end = __rdtsc(); | | | | | | | |
| 526 | sum_foo += end - start; | | | | | | | |
| 527 | start = __rdtsc(); | | | | | | | |
| 528 | for (int l = 0; l < nLoops; l++) | | | | | | | |
| 529 | { | | | | | | | |
| 530 | for (int i = 0; i < sizeof(interested_ids) / | 0.001s | 0.001s | 0.0% | 0.0% | 0 | 0.0% | 0 |
| 531 | { | | | | | | | |
| 532 | | | | | | | | |
| 533 | goo = GetGooByID(g_GooData, sizeof(g_Goo | 1.544s | 1.544s | 0.0% | 0.0% | 48.2% | 2,043,661... | 0.0% |
| 534 | if (goo) | 0.033s | 0.033s | 0.0% | 0.0% | 1.0% | 40,801,224 | 0.0% |
| 535 | extern_ref += goo->szName_[0]; | | | | | | | |
| 536 | } | | | | | | | |
| 537 | } | | | | | | | |
| 538 | end = __rdtsc(); | | | | | | | |
| 539 | sum_goo += end - start; | | | | | | | |

gecode

r000ue Analysis Configuration Collection Log Summary Event Count Sample Count Uncore Event Count Caller/Callee Top-down Tree Platform clabs.c ×

Source

Assembly



r002ue

r003ue

r004ue

r005ue

r006ue

r007...

r008...

r009...

r010...

Source

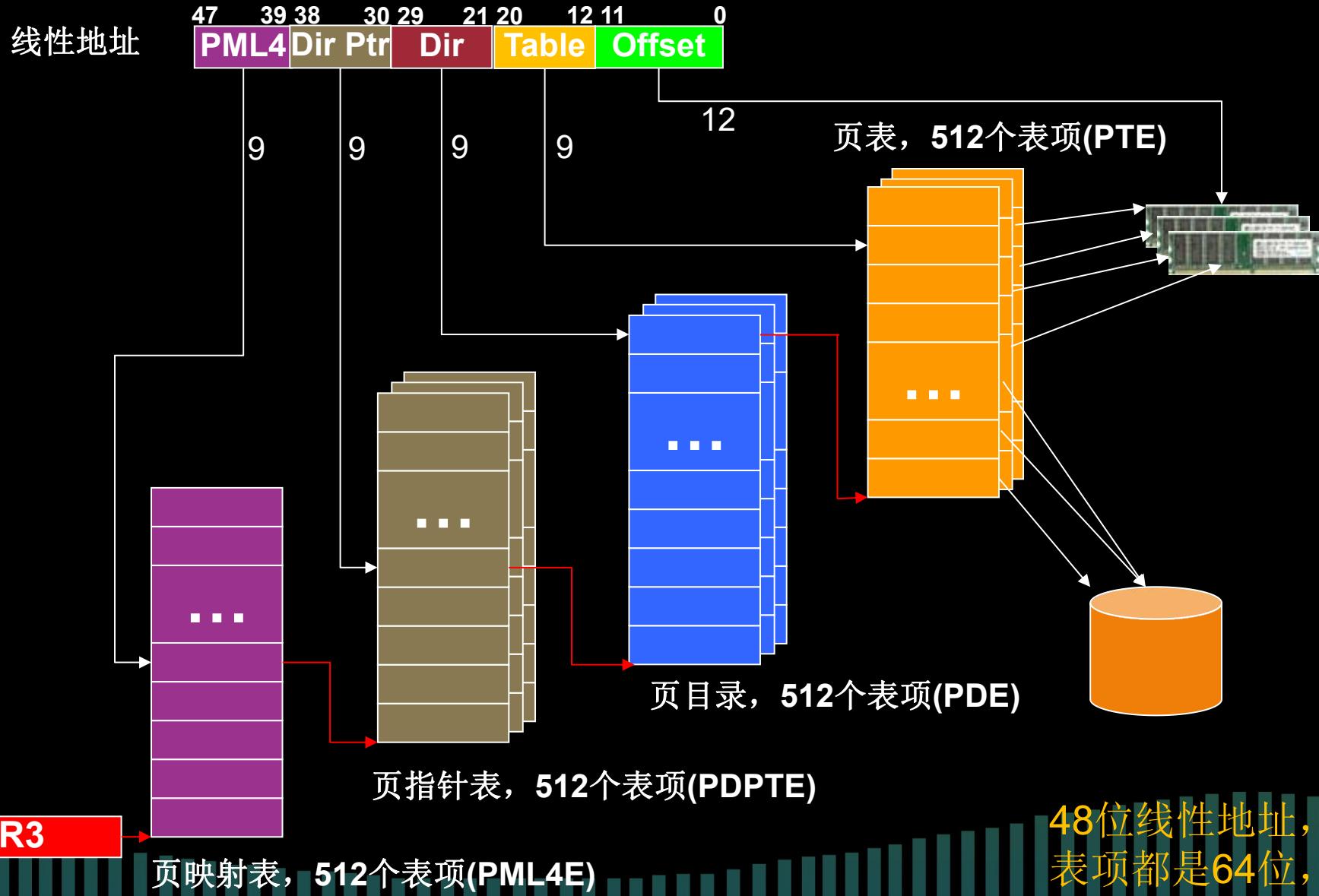
```

511     long long unsigned int sum_foo = 0, sum_goo = 0;
512     long extern_ref = 0;
513     GOO_DATA_S* goo;
514     FOO_DATA_S* foo;
515     start = __rdtsc();
516     for (int l = 0; l < nLoops; l++)
517     {
518         for (int i = 0; i < sizeof(interested_ids) / sizeof(int)
519         {
520             foo = GetFooByID(g_FooData, sizeof(g_FooData) / si
521             if (foo)
522                 extern_ref += foo->szName_[0];
523         }
524     }
525     end = __rdtsc();
526     sum_foo += end - start;
527     start = __rdtsc();
528     for (int l = 0; l < nLoops; l++)
529     {
530         for (int i = 0; i < sizeof(interested_ids) / sizeof(int)
531         {
532             goo = GetGooByID(g_GooData, sizeof(g_GooData) / si
533             if (goo)
534                 extern_ref += goo->szName_[0];
535         }
536     }
537     end = __rdtsc();
538     sum_goo += end - start;
539     printf("GeLabIntactBatch: %d loops is done\n"

```

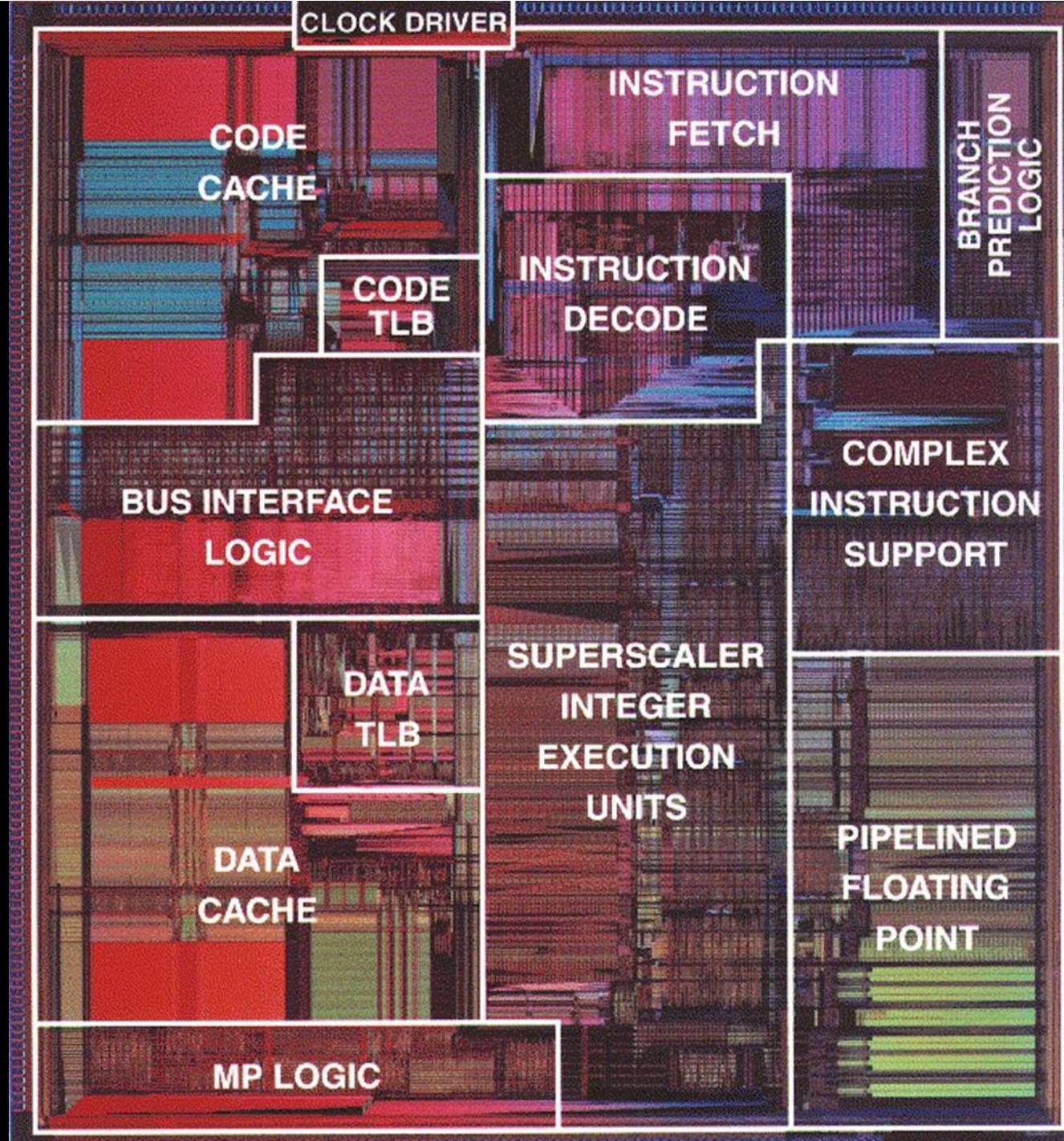
| | LLS_MEM_ANY | DTLB_LOAD_MISSES.STLB_HIT | DTLB_LOAD_MISSES.WALK_ACTIVE | DTLB_STORE_MISSES.STLB_HIT | DT |
|-----|-------------|---------------------------|------------------------------|----------------------------|----|
| 511 | 0 | 0 | 0 | 0 | 0 |
| 512 | 0 | 0 | 46,801,404 | 0 | 0 |
| 513 | 0 | 0 | 0 | 0 | 0 |
| 514 | 0 | 0 | 0 | 0 | 0 |
| 515 | 0 | 0 | 0 | 0 | 0 |
| 516 | 0 | 0 | 0 | 0 | 0 |
| 517 | 0 | 0 | 0 | 0 | 0 |
| 518 | 0 | 0 | 0 | 0 | 0 |
| 519 | 0 | 0 | 0 | 0 | 0 |
| 520 | 0 | 46,801,404 | 0 | 0 | 0 |
| 521 | 0 | 0 | 0 | 0 | 0 |
| 522 | 0 | 0 | 0 | 0 | 0 |
| 523 | 0 | 0 | 0 | 0 | 0 |
| 524 | 0 | 0 | 0 | 0 | 0 |
| 525 | 0 | 0 | 0 | 0 | 0 |
| 526 | 0 | 0 | 0 | 0 | 0 |
| 527 | 0 | 0 | 0 | 0 | 0 |
| 528 | 0 | 0 | 0 | 0 | 0 |
| 529 | 0 | 0 | 0 | 0 | 0 |
| 530 | 0 | 0 | 0 | 0 | 0 |
| 531 | 0 | 0 | 0 | 0 | 0 |
| 532 | 0 | 0 | 0 | 0 | 0 |
| 533 | 0 | 0 | 0 | 0 | 0 |
| 534 | 0 | 0 | 0 | 0 | 0 |
| 535 | 0 | 0 | 0 | 0 | 0 |
| 536 | 0 | 0 | 0 | 0 | 0 |
| 537 | 0 | 0 | 0 | 0 | 0 |
| 538 | 0 | 0 | 0 | 0 | 0 |
| 539 | 0 | 0 | 0 | 0 | 0 |
| 540 | 0 | 0 | 0 | 0 | 0 |

地址翻译 (64位, 4KB页)



TLB

- Translation Lookaside Buffer
- a small associative memory that caches virtual to physical page table resolutions



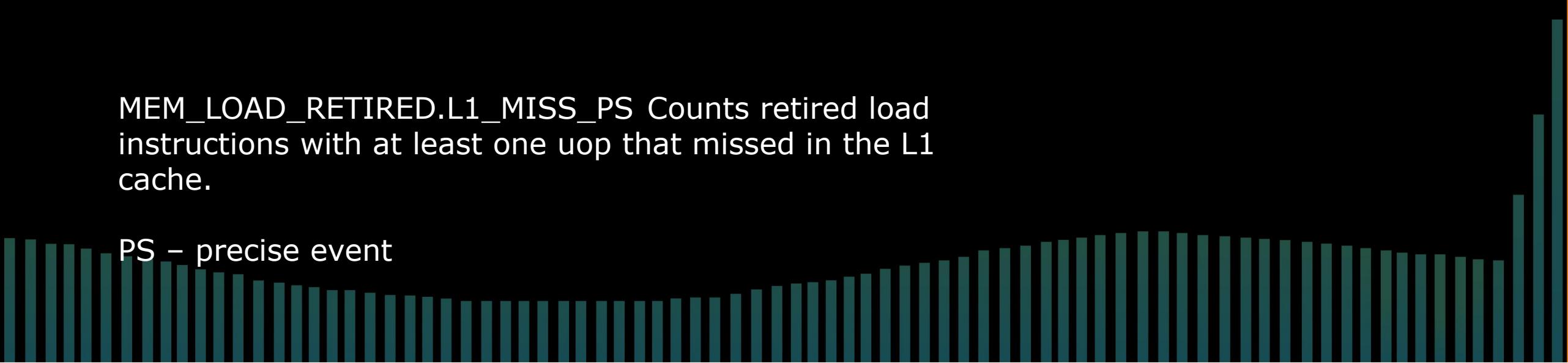
Ubuntu on GDK7的数据

Grouping: Function / Call Stack

| Function / Call Stack | MEM LOAD RETIRED.L1 MISS PS | MEM LOAD RETIRED.L2 HIT PS | MEM LOAD RETIRED.L3 HIT PS | MEM LOAD |
|----------------------------|-----------------------------|----------------------------|----------------------------|----------|
| | | | | |
| ► GeLabIntactBatchUnlikely | 0 | 0 | 0 | |
| ► GetGooByIDWithUnlikely | 0 | 0 | 0 | |
| ► GetFooByIDWithUnlikely | 4,800,144 | 2,400,072 | 600,252 | |

MEM_LOAD_RETIRED.L1_MISS_PS Counts retired load instructions with at least one uop that missed in the L1 cache.

PS - precise event





C君直言3：

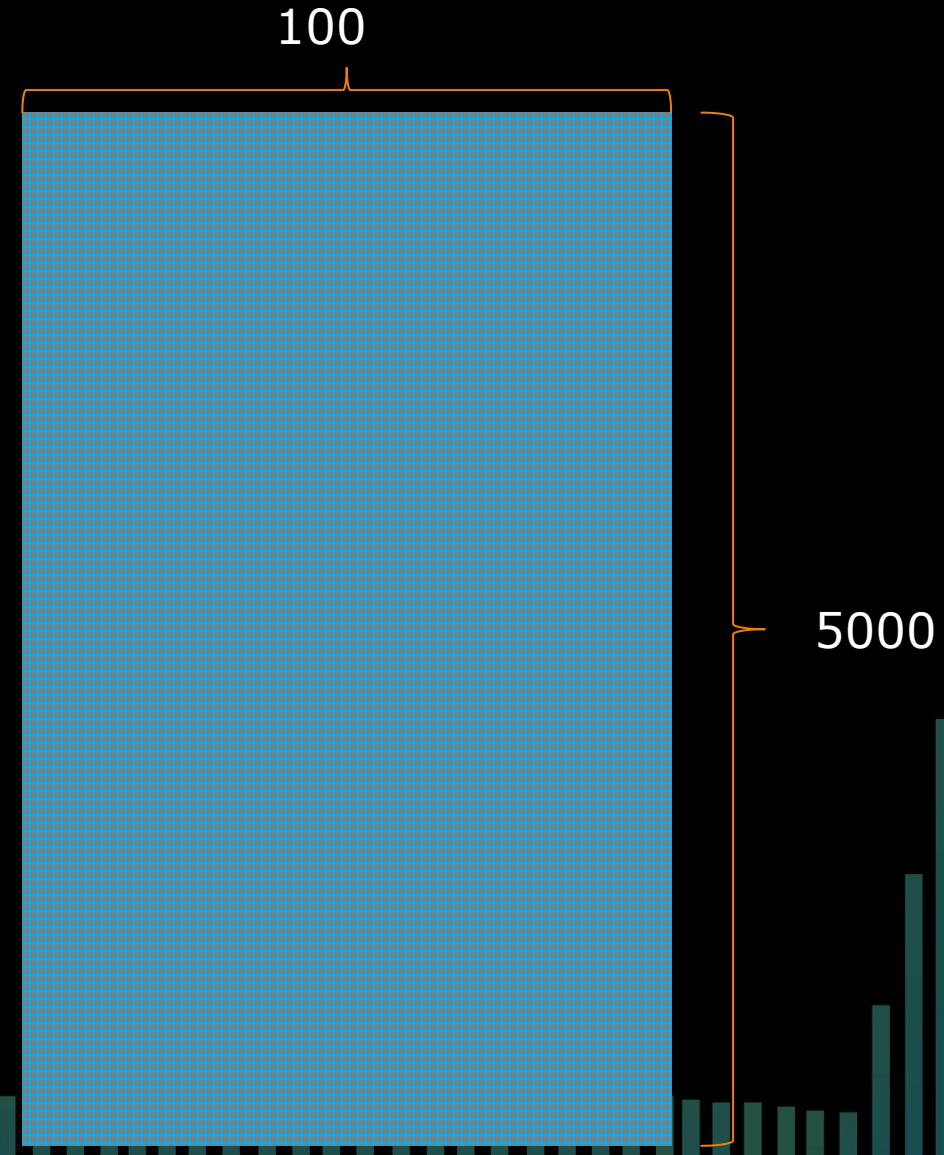
牛糞似的松散结构体不仅浪费内存，
而且访问的效率很低

X[5000][100]

问题4：

要把右侧的二维矩阵中的每个元素翻倍，以下两种写法哪个速度快？

```
/* A */  
    for (j = 0; j < 100; j = j+1)  
        for (i = 0; i < 5000; i = i+1)  
            x[i][j] = 2 * x[i][j];  
  
/* B */  
    for (i = 0; i < 5000; i = i+1)  
        for (j = 0; j < 100; j = j+1)  
            x[i][j] = 2 * x[i][j];
```



```
29 uint64_t ge_matrix_sum(int loops)
30 {
31     uint64_t start, end, sum = 0;
32
33     /* A */
34     start = __rdtsc();
35     for (int l = 0; l < loops; l++)
36     {
37         for (int j = 0; j < ARRAY_MAX_COL; j = j + 1)
38             for (int i = 0; i < ARRAY_MAX_ROW; i = i + 1)
39                 sum += g_matrix_a[i][j];
40     }
41     end = __rdtsc();
42     cout << "cross row scan took " << end - start << " ticks. aver " << (end - start) / loops << endl;
43     /* B */
44     start = __rdtsc();
45     for (int l = 0; l < loops; l++)
46     {
47         for (int i = 0; i < ARRAY_MAX_ROW; i = i + 1)
48             for (int j = 0; j < ARRAY_MAX_COL; j = j + 1)
49                 sum += g_matrix_b[i][j];
50     }
51     end = __rdtsc();
52     cout << "row by row scan took " << end - start << " ticks. aver " << (end - start) / loops << endl;
53
54     return sum;
55 }
```

```
cross row scan took 719991163 ticks. aver 719991  
row by row scan took 689074117 ticks. aver 689074
```

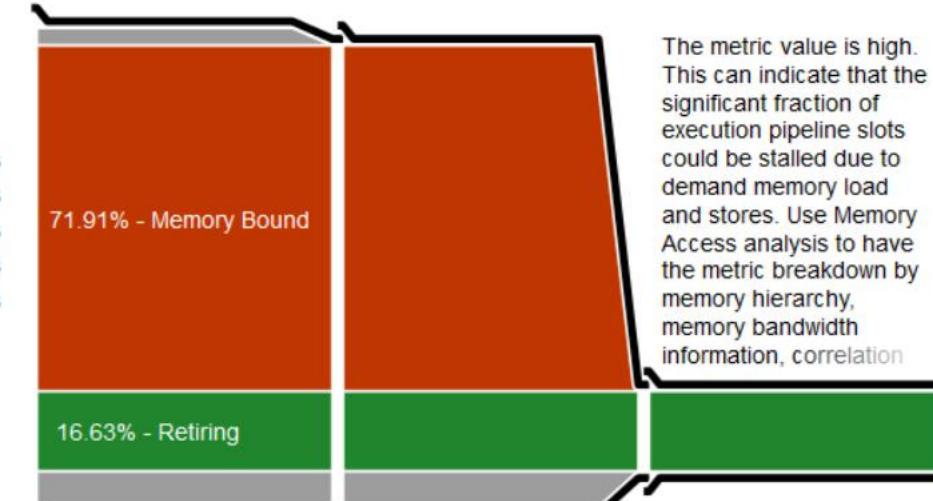
```
0:000> ?? (719991-689074)/719991.0  
double 0.042940814537959504094
```

Elapsed Time ?: 74.106s

| | |
|--|--|
| Clockticks: | 949,896,000,000 |
| Instructions Retired: | 312,003,000,000 |
| CPI Rate ? : | 3.045 ? |
| MUX Reliability ? : | 0.997 |
| Retiring ? : | 16.6% of Pipeline Slots |
| Front-End Bound ? : | 3.7% of Pipeline Slots |
| Bad Speculation ? : | 0.2% of Pipeline Slots |
| Back-End Bound ? : | 79.5% ? of Pipeline Slots |
| Memory Bound ? : | 71.9% ? of Pipeline Slots |
| L1 Bound ? : | 2.9% ? of Clockticks |
| DTLB Overhead ? : | 22.5% ? of Clockticks |
| Loads Blocked by Store Forwarding ? : | 0.0% of Clockticks |
| Lock Latency ? : | 0.0% ? of Clockticks |
| Split Loads ? : | 0.0% of Clockticks |
| 4K Aliasing ? : | 0.2% of Clockticks |
| FB Full ? : | 0.0% ? of Clockticks |
| L2 Bound ? : | 1.1% of Clockticks |
| L3 Bound ? : | 8.2% ? of Clockticks |
| DRAM Bound ? : | 68.2% ? of Clockticks |
| Memory Bandwidth ? : | 44.1% ? of Clockticks |
| Memory Latency ? : | 48.7% ? of Clockticks |
| LLC Miss ? : | 100.0% ? of Clockticks |
| Store Bound ? : | 0.0% of Clockticks |
| Core Bound ? : | 7.6% of Pipeline Slots |
| Divider ? : | 0.0% of Clockticks |
| Port Utilization ? : | 8.5% of Clockticks |
| Cycles of 0 Ports Utilized ? : | 64.5% of Clockticks |
| Cycles of 1 Port Utilized ? : | 13.0% of Clockticks |
| Cycles of 2 Ports Utilized ? : | 11.9% of Clockticks |
| Cycles of 3+ Ports Utilized ? : | 10.3% of Clockticks |
| Vector Capacity Usage (FPU) ? : | 25.0% ? |

Total Thread Count:

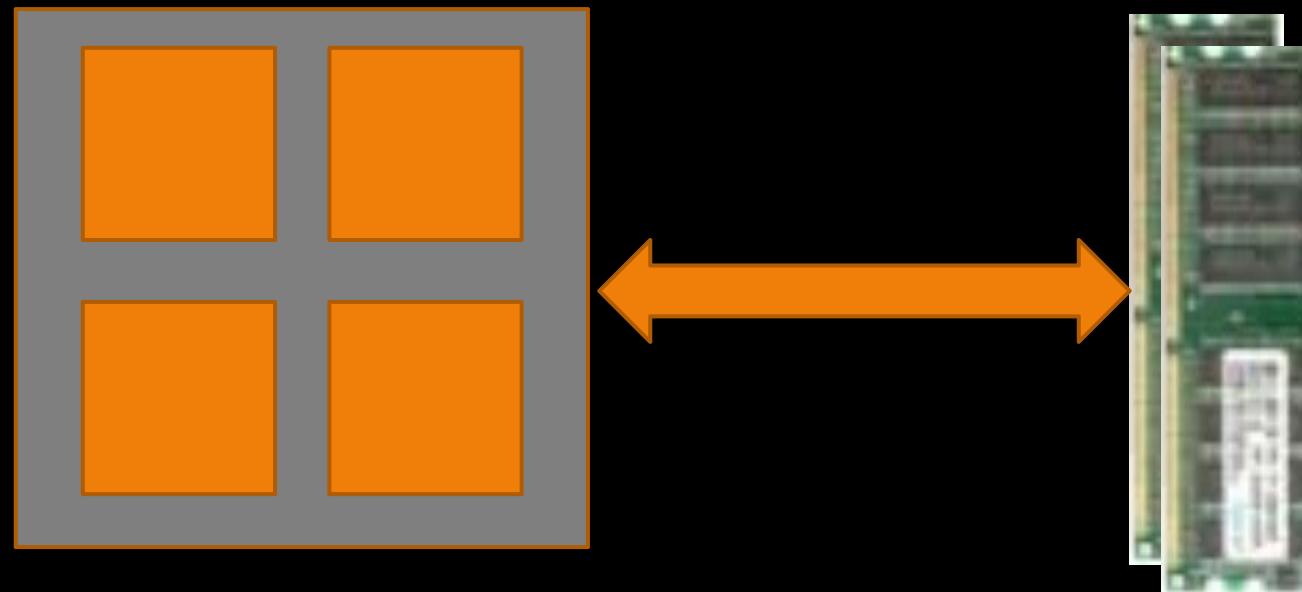
10

**μPipe**

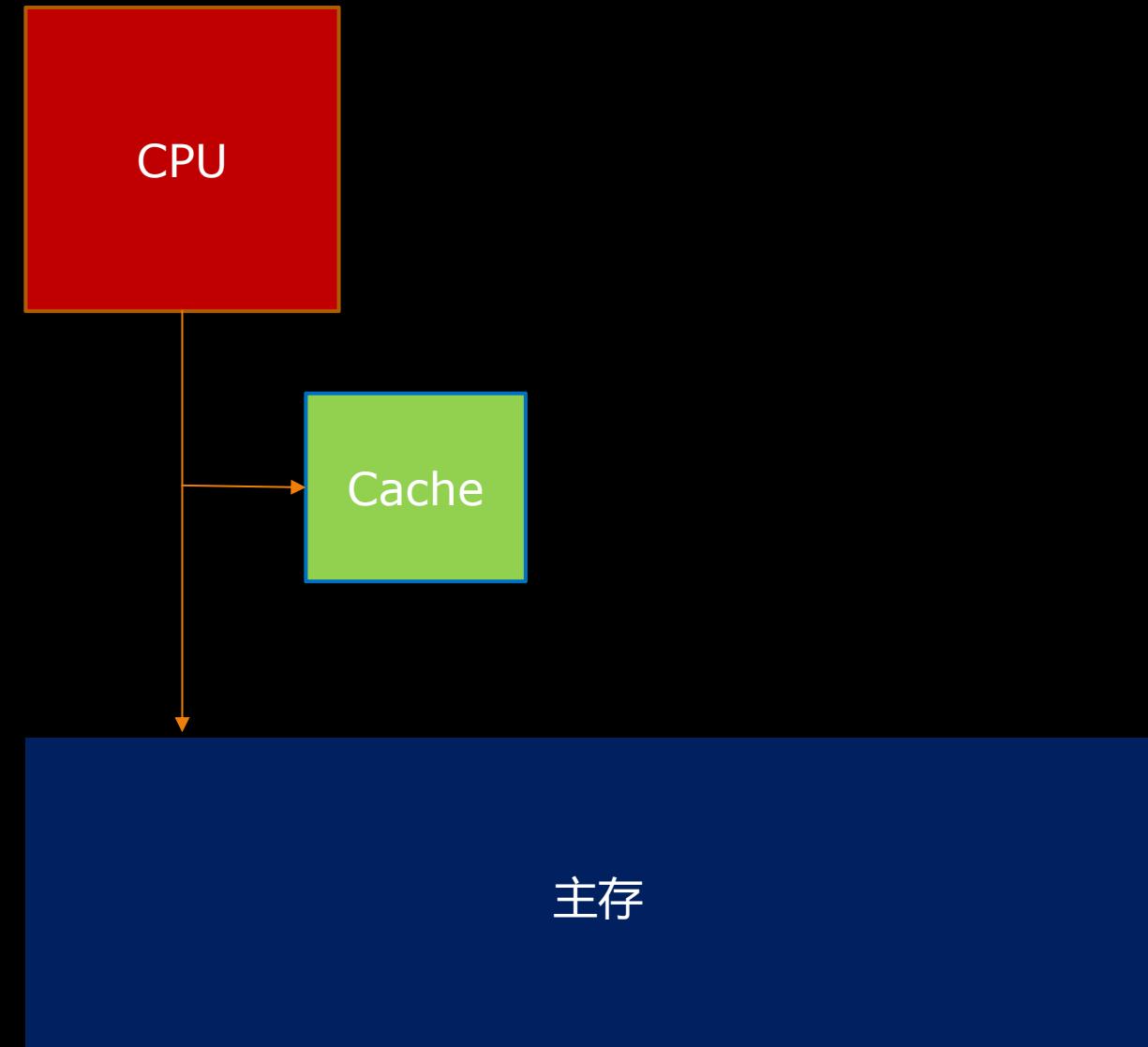
This diagram represents inefficiencies in CPU usage. Treat it as a pipe with an output flow equal to the "pipe efficiency" ratio: (Actual Instructions Retired)/(Maximum Possible Instruction Retired). If there are pipeline stalls decreasing the pipe efficiency, the pipe shape gets more narrow.

内存限制是制约性能的最常见因素

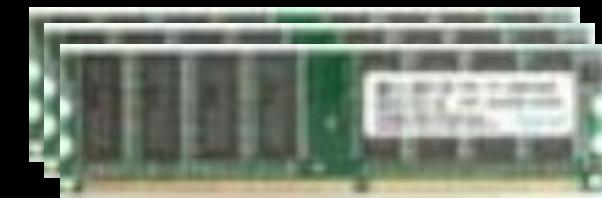
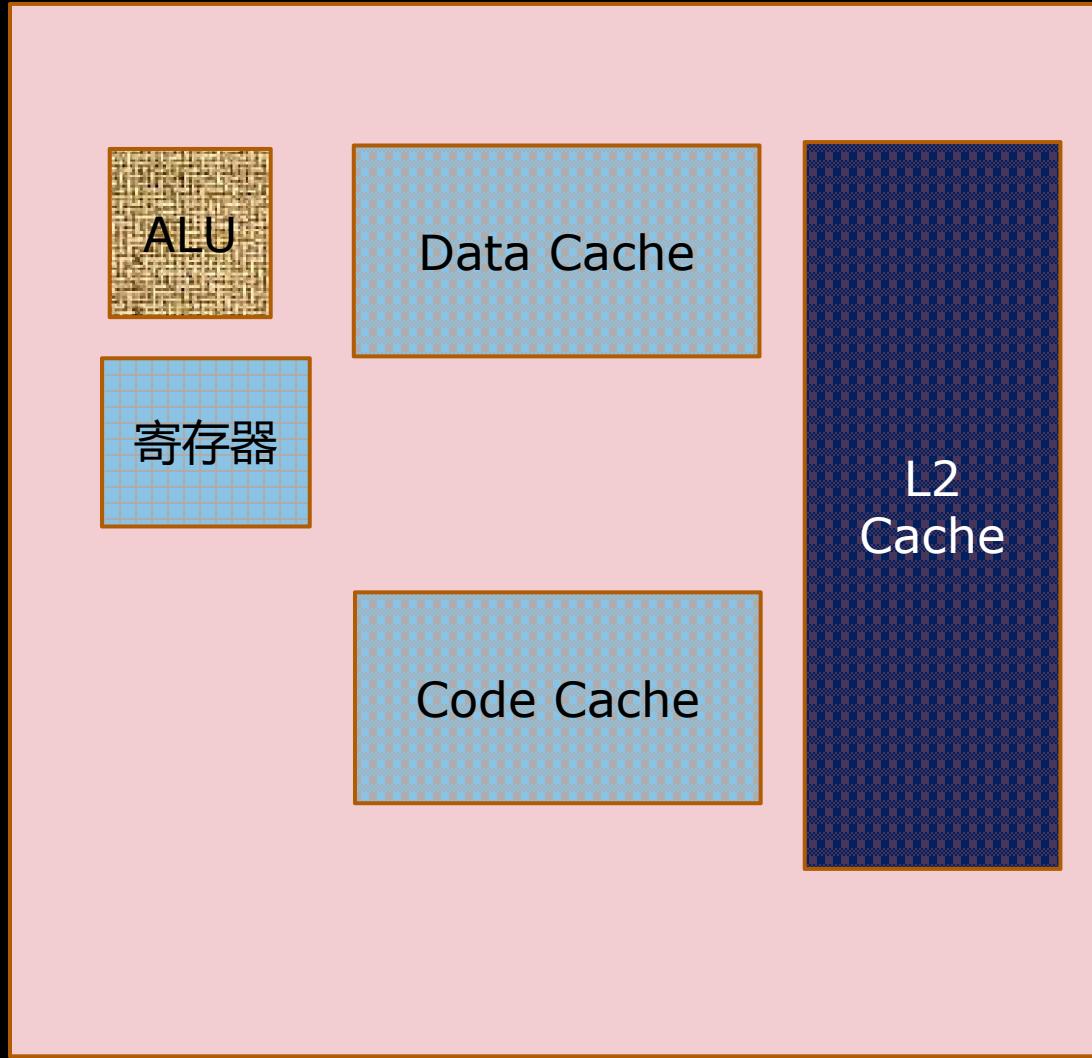
多核化加重了内存约束



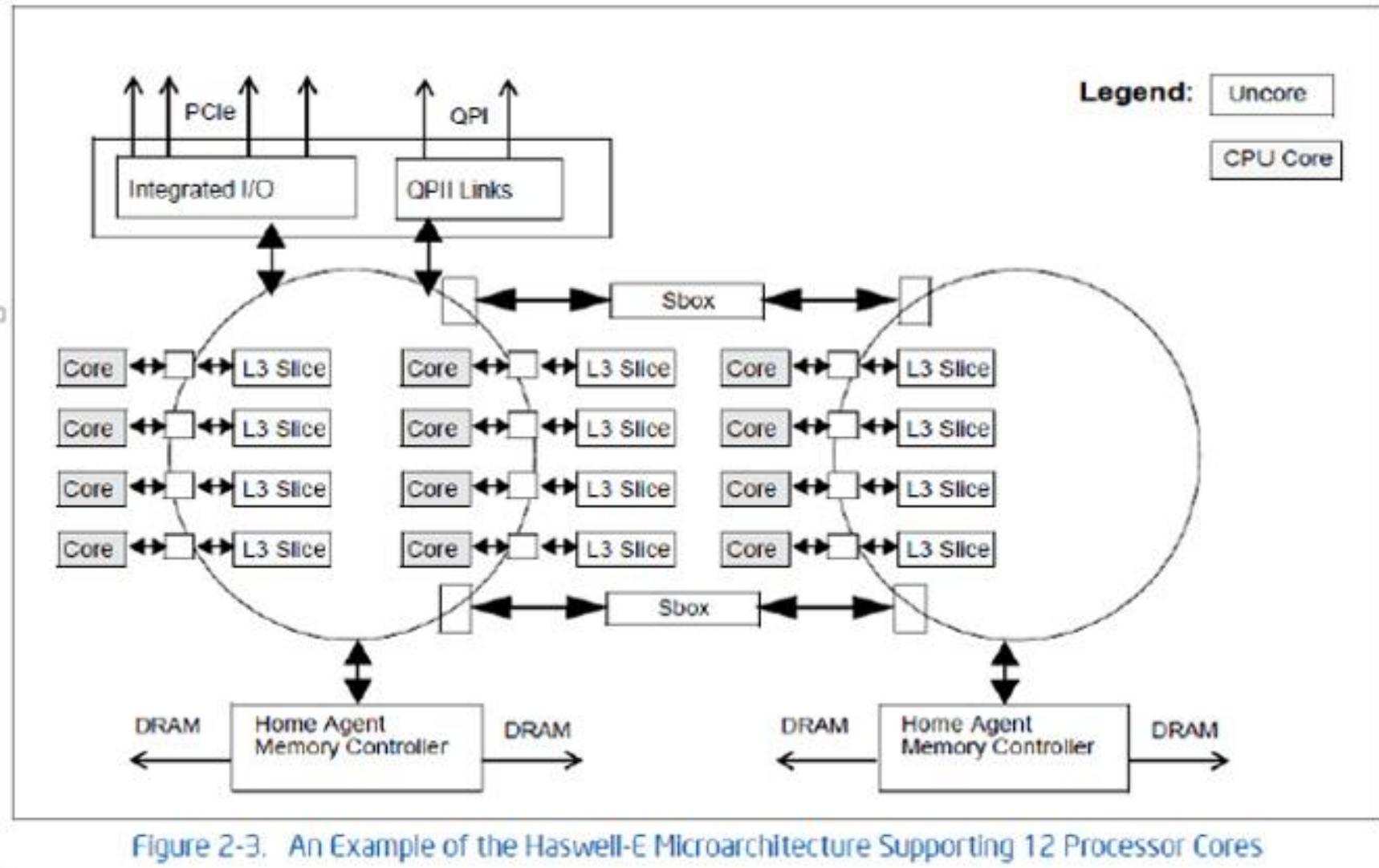
解决之道



ALU/CPU的储物箱



E5 Xeon Architecture





Intel® Xeon® Platinum 8380HL Processor

[Add to Compare](#)

38.5M Cache, 2.90 GHz

Specifications

Essentials

[CPU Specifications](#)[Supplemental Information](#)[Memory Specifications](#)[Expansion Options](#)[Package Specifications](#)[Advanced Technologies](#)[Security & Reliability](#)[Ordering and Compliance](#)

Essentials

 [Export specifications](#)

Product Collection

3rd Generation Intel® Xeon® Scalable Processors

Code Name

Products formerly Cooper Lake

Vertical Segment

Server

Processor Number

8380HL

Status

Launched

Launch Date

Q2'20

Recommended Customer Price

\$13012.00



循环交换

```
void multiply2(int msize, int tidx, int numt, TYPE a[][NUM], TYPE
b[][NUM], TYPE c[][NUM], TYPE t[][NUM])
{
    int i,j,k;

// Step 2: Loop interchange
    for(i=tidx; i<msize; i=i+numt) {
        for(k=0; k<msize; k++) {
            for(j=0; j<msize; j++) {
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
            }
        }
    }
}
```

- 编译执行，记录时间
- 数量级提升

合并循环

```
/* Before */  
for (i = 0; i < N; i = i+1)  
    for (j = 0; j < N; j = j+1)  
        a[i][j] = 1/b[i][j] * c[i][j];  
for (i = 0; i < N; i = i+1)  
    for (j = 0; j < N; j = j+1)  
        d[i][j] = a[i][j] + c[i][j];  
  
/* After */  
for (i = 0; i < N; i = i+1)  
    for (j = 0; j < N; j = j+1)  
    {  
        a[i][j] = 1/b[i][j] * c[i][j];  
        d[i][j] = a[i][j] + c[i][j];  
    }
```

- 减少cache miss



C君直言4:

Cache多少代表了我的身价，用不好
Cache就是白白浪费了我的最宝贵资源



编写Cache友好代码的第一原则：局部性原则

授人以渔



第一次看效果

```
(gdb) r
Starting program: /mnt/e/cplabs/gecache/out/build/WSL-Release/gecache 1 1000000
GeCache [labno] [loops] rev2 by Raymond.
Cache lab to compare struct with names
GeLabIntact: 1000000 loops is done
    Nonintact struct total 117954026 ticks aver 117 ticks
    Intact   struct total 117291580 ticks aver 117 ticks
[Inferior 1 (process 2816) exited normally]
```

没有差异



纳秒级调优的第一大挑战是常常看不出效果或者看到相反的效果

```
int GeLabIntactInterlaceNoRef(int nLoops)
{
    printf("Cache lab to compare struct with names\n");
    int interesed_ids[] = { 1,101,188,2,5,88,66,155,133 };
    long long unsigned int start, end;
    long long unsigned int sum_foo = 0, sum_goo = 0;
    GOO_DATA_S* goo;
    FOO_DATA_S* foo;
    for (int l = 0; l < nLoops; l++)
    {
        for (int i = 0; i < sizeof(interesed_ids) / sizeof(interesed_ids[0]); i++)
        {
            start = __rdtsc();
            goo = GetGooByID(g_GooData, sizeof(g_GooData) / sizeof(g_GooData[0]), interesed_ids[i]);
            end = __rdtsc();
            sum_goo += end - start;

            start = __rdtsc();
            foo = GetFooByID(g_FooData, sizeof(g_FooData) / sizeof(g_FooData[0]), interesed_ids[i]);
            end = __rdtsc();
            sum_foo += end - start;
        }
    }
    printf("GeLabIntact: %d loops is done\n"
        "\tNonintact struct total %llu ticks aver %llu ticks\n"
        "\tIntact    struct total %llu ticks aver %llu ticks\n",
        nLoops, sum_foo, sum_foo / nLoops, sum_goo, sum_goo / nLoops
    );
}
```

请调试器帮忙

```
(gdb) disassemble GeLabIntact
Dump of assembler code for function GeLabIntact:
0x00000000080012e0 <+0>:    push   %rbx
0x00000000080012e1 <+1>:    mov    %edi, %ebx
0x00000000080012e3 <+3>:    lea    0x176(%rip), %rdi      # 0x8001460
0x00000000080012ea <+10>:   callq  0x8001060 <puts@plt>
0x00000000080012ef <+15>:   test   %ebx, %ebx
0x00000000080012f1 <+17>:   jle    0x8001388 <GeLabIntact+168>
0x00000000080012f7 <+23>:   movslq %ebx, %r10
0x00000000080012fa <+26>:   xor    %r8d, %r8d
0x00000000080012fd <+29>:   xor    %r9d, %r9d
0x0000000008001300 <+32>:   xor    %ecx, %ecx
0x0000000008001302 <+34>:   nopw   0x0(%rax, %rax, 1)
0x0000000008001308 <+40>:   mov    $0x9, %edi
0x000000000800130d <+45>:   nopl   (%rax)
0x0000000008001310 <+48>:   rdtsc 
0x0000000008001312 <+50>:   mov    %rax, %rsi
0x0000000008001315 <+53>:   shl    $0x20, %rdx
0x0000000008001319 <+57>:   or     %rdx, %rsi
0x000000000800131c <+60>:   rdtsc 
0x000000000800131e <+62>:   shl    $0x20, %rdx
0x0000000008001322 <+66>:   or     %rdx, %rax
0x0000000008001325 <+69>:   sub    %rsi, %rax
0x0000000008001328 <+72>:   add    %rax, %r9
0x000000000800132b <+75>:   rdtsc 
0x000000000800132d <+77>:   mov    %rax, %rsi
0x0000000008001330 <+80>:   shl    $0x20, %rdx
0x0000000008001334 <+84>:   or     %rdx, %rsi
0x0000000008001337 <+87>:   rdtsc 
0x0000000008001339 <+89>:   shl    $0x20, %rdx
---Type <return> to continue, or q <return> to quit---
0x000000000800133d <+93>:   or     %rdx, %rax
0x0000000008001340 <+96>:   sub    %rsi, %rax
0x0000000008001343 <+99>:   add    %rax, %rcx
0x0000000008001346 <+102>:  sub    $0x1, %edi
0x0000000008001349 <+105>:  jne    0x8001310 <GeLabIntact+48>
0x000000000800134b <+107>:  add    $0x1, %r8d
0x000000000800134f <+111>:  cmp    %r8d, %ebx
0x0000000008001352 <+114>:  jne    0x8001308 <GeLabIntact+40>
0x0000000008001354 <+116>:  mov    %r9, %rax
0x0000000008001357 <+119>:  xor    %edx, %edx
0x0000000008001359 <+121>:  div    %r10
0x000000000800135c <+124>:  xor    %edx, %edx
0x000000000800135e <+126>:  mov    %rax, %rsi
```

```
232 int GeLabIntact(int nLoops)
233 {
234     printf("Cache lab to compare struct with names\n");
235     int interesed_ids[] = { 1,101,188,2,5,88,66,155,133 };
236     long long unsigned int start, end;
237     long long unsigned int sum_foo = 0, sum_goo = 0;
238     long extern_ref = 0;
239     GOO_DATA_S* goo;
240     FOO_DATA_S* foo;
241     for(int l =0;l< nLoops;l++)
242     {
243         for (int i = 0; i < sizeof(interesed_ids)/sizeof(interesed_ids[0]); i++)
244         {
245             start = __rdtsc();
246             goo = GetGooByID(g_GooData, sizeof(g_GooData) / sizeof(g_GooData[0]), interesed_ids[i]);
247             if (goo)
248                 extern_ref += goo->szName_[0];
249             end = __rdtsc();
250             sum_goo += end - start;
251
252             start = __rdtsc();
253             foo = GetFooByID(g_FooData, sizeof(g_FooData) / sizeof(g_FooData[0]), interesed_ids[i]);
254             if (foo)
255                 extern_ref += foo->szName_[0];
256             end = __rdtsc();
257             sum_foo += end - start;
258         }
259     }
260     printf("GeLabIntact: %d loops is done\n"
261           "\tNonintact struct total %llu ticks aver %llu ticks\n"
262           "\tIntact    struct total %llu ticks aver %llu ticks\n",
263           nLoops, sum_foo, sum_foo/ nLoops, sum_goo, sum_goo/ nLoops
264     );
265     return extern_ref; // to avoid code cutting by compiler
266 }
267
268 }
```

增加输出，
防止编译
器好心做
坏事

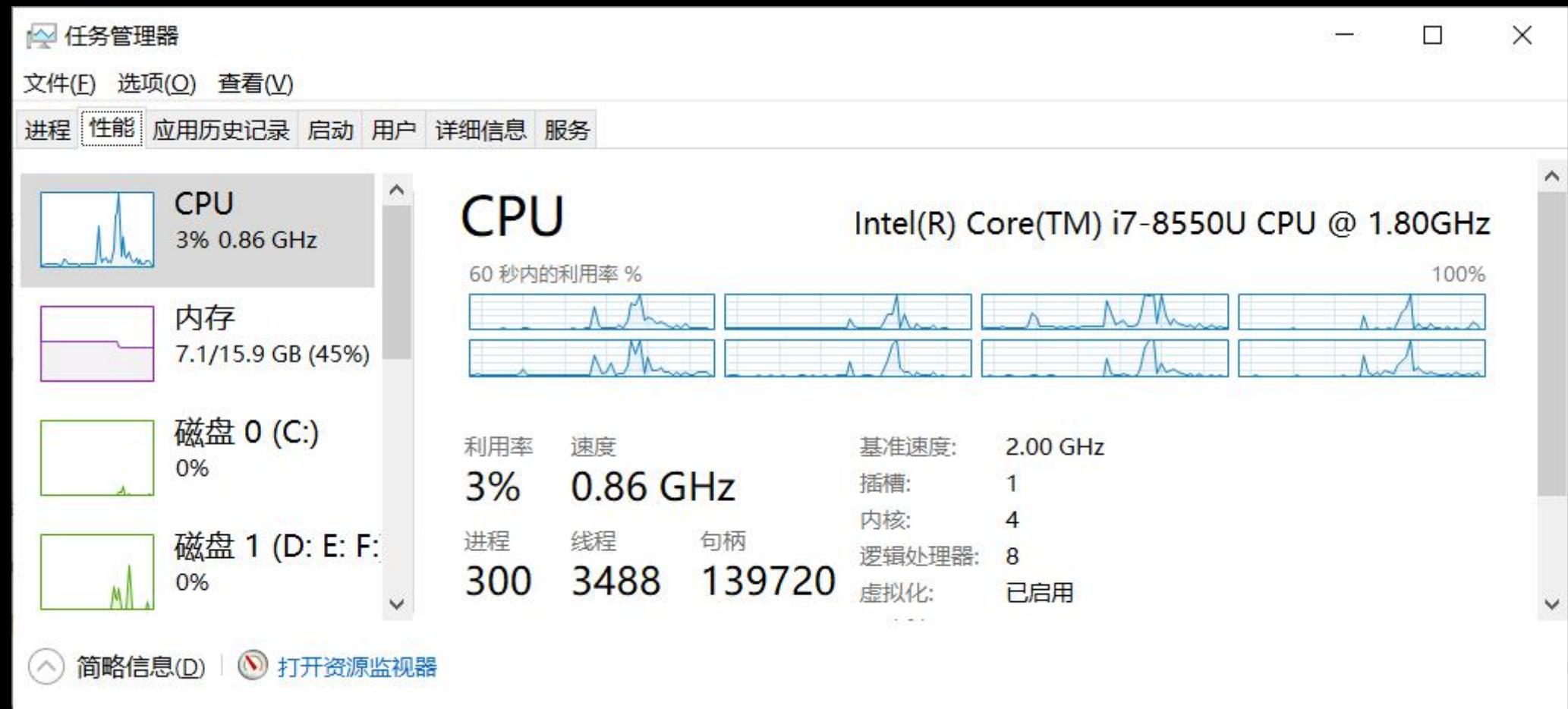
```

0x00000000000000013d6 <+134>: lea    0x21a087(%rip),%rdi      # 0x21b404
0x00000000000000013dd <+141>: lea    0x24(%r14),%r12
0x00000000000000013e1 <+145>: xor   %r13d,%r13d
0x00000000000000013e4 <+148>: xor   %ecx,%ecx
0x00000000000000013e6 <+150>: xor   %r9d,%r9d
0x00000000000000013e9 <+153>: xor   %ebx,%ebx
0x00000000000000013eb <+155>: lea    0x4(%r14),%r11
0x00000000000000013ef <+159>: mov    $0x1,%esi
0x00000000000000013f4 <+164>: nopl  0x0(%rax)
0x00000000000000013f8 <+168>: rdtsc
0x00000000000000013fa <+170>: mov    %rax,%r10
0x00000000000000013fd <+173>: shl   $0x20,%rdx
0x0000000000000001401 <+177>: lea    0x21a078(%rip),%rax      # 0x21b480 <g_GooData>
0x0000000000000001408 <+184>: or    %rdx,%r10
0x000000000000000140b <+187>: jmp   0x1419 <GeLabIntact+201>
0x000000000000000140d <+189>: nopl  (%rax)
0x0000000000000001410 <+192>: add   $0x10,%rax
0x0000000000000001414 <+196>: cmp   %rax,%r8
0x0000000000000001417 <+199>: je    0x1428 <GeLabIntact+216>
0x0000000000000001419 <+201>: cmp   (%rax),%esi
0x000000000000000141b <+203>: jne   0x1410 <GeLabIntact+192>
>Type <return> to continue, or q <return> to quit---
0x000000000000000141d <+205>: mov   0x8(%rax),%rax
0x0000000000000001421 <+209>: movsbq (%rax),%rax
0x0000000000000001425 <+213>: add   %rax,%rcx
0x0000000000000001428 <+216>: rdtsc
0x000000000000000142a <+218>: shl   $0x20,%rdx
0x000000000000000142e <+222>: or    %rdx,%rax
0x0000000000000001431 <+225>: sub   %r10,%rax
0x0000000000000001434 <+228>: add   %rax,%r9
0x0000000000000001437 <+231>: rdtsc
0x0000000000000001439 <+233>: mov   %rax,%r10
0x000000000000000143c <+236>: shl   $0x20,%rdx
0x0000000000000001440 <+240>: lea    0x201bd9(%rip),%rax      # 0x203020 <g_FooData>
0x0000000000000001447 <+247>: or    %rdx,%r10
0x000000000000000144a <+250>: jmp   0x145b <GeLabIntact+267>
0x000000000000000144c <+252>: nopl  0x0(%rax)
0x0000000000000001450 <+256>: add   $0x3ec,%rax
0x0000000000000001456 <+262>: cmp   %rax,%rdi
0x0000000000000001459 <+265>: je    0x1467 <GeLabIntact+279>
0x000000000000000145b <+267>: cmp   (%rax),%esi
0x000000000000000145d <+269>: jne   0x1450 <GeLabIntact+256>
0x000000000000000145f <+271>: movsbq 0x4(%rax),%rax
0x0000000000000001464 <+276>: add   %rax,%rcx

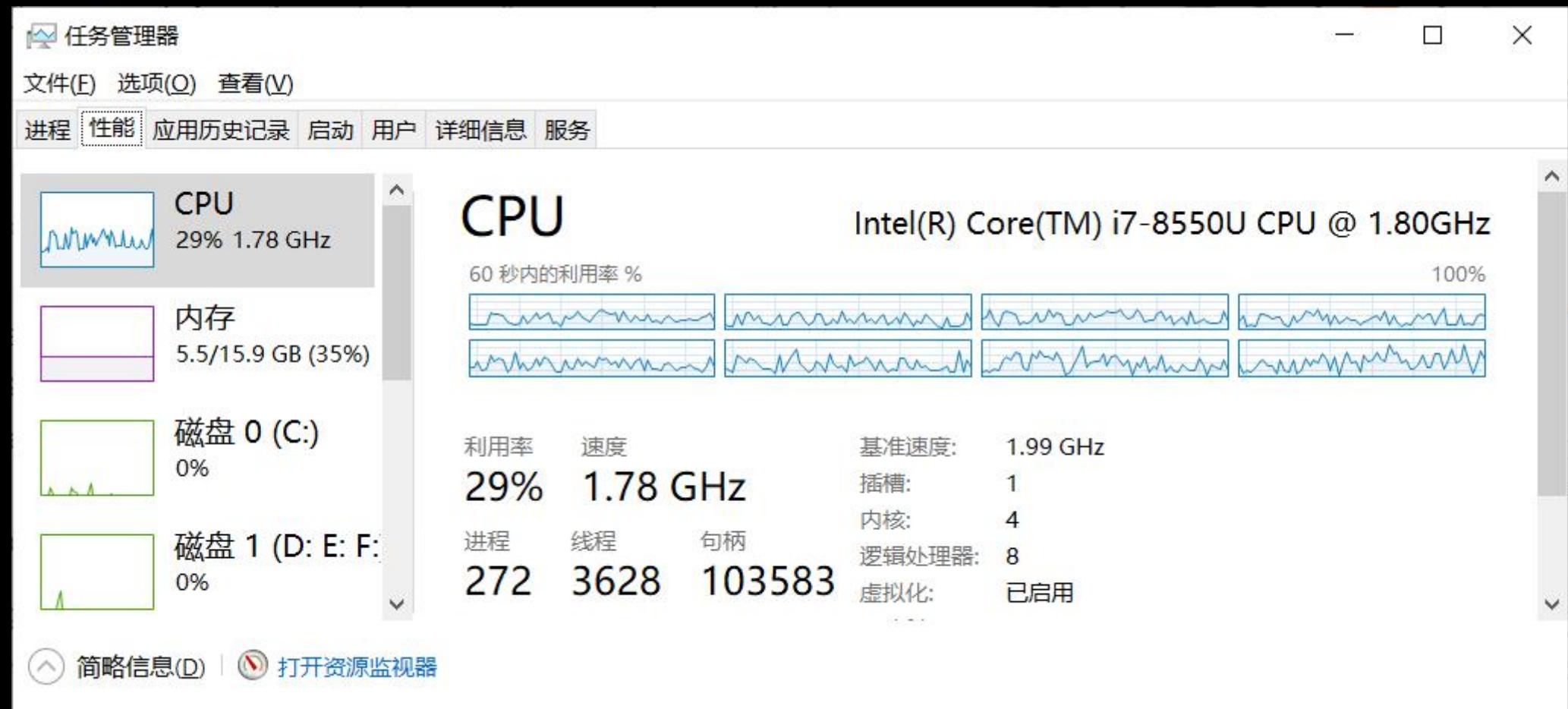
```

自动inline了

纳秒级调优第一守则：防 止编译器好心帮倒忙

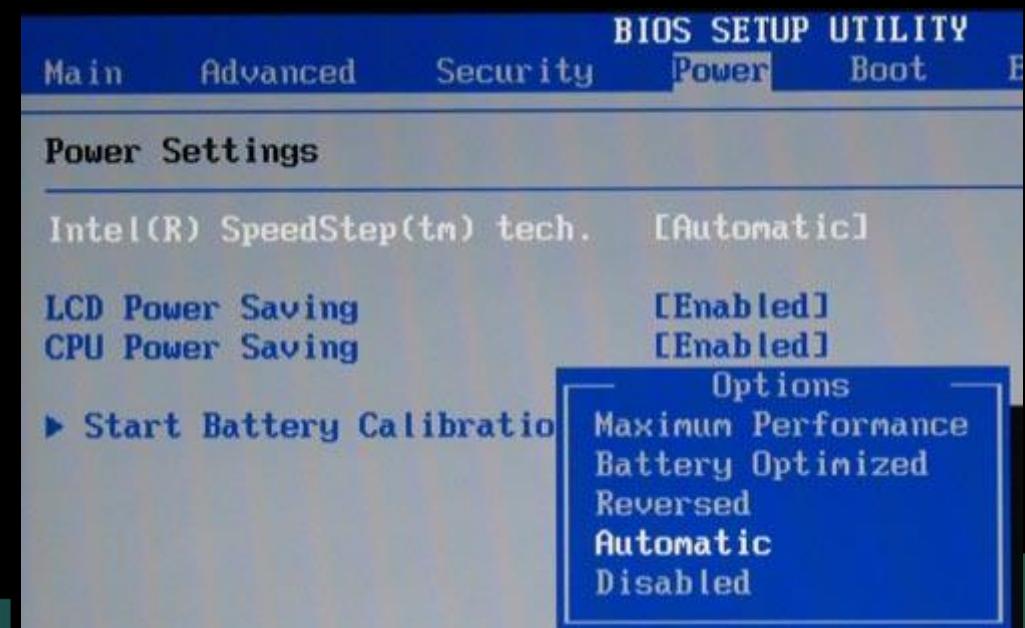


纳秒级别调优，有很多干扰因素，让我们迷失，Speed Step技术是其中一个



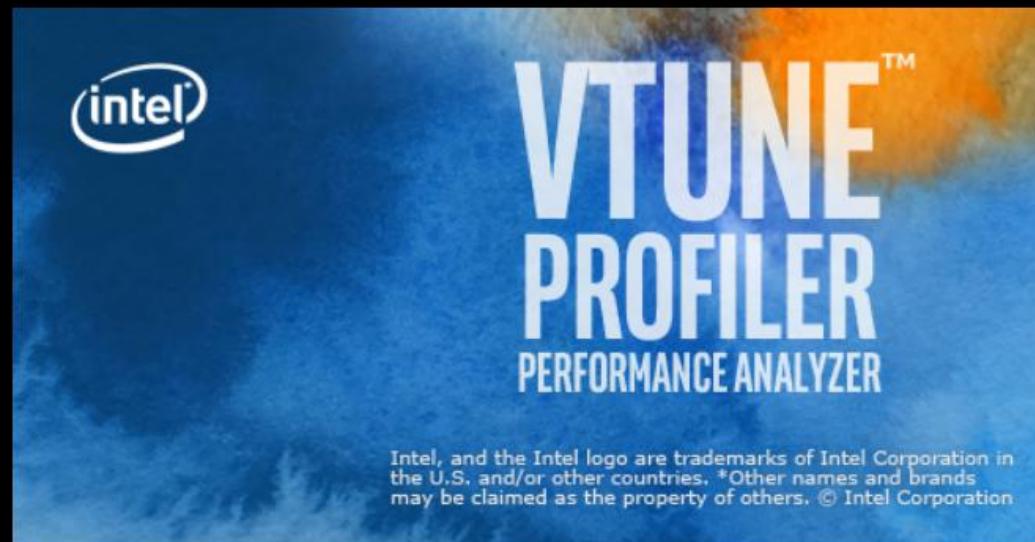
纳秒级调优第二守则：禁止CPU调频

优化得到的效果常常抵不上频率成倍跳动产生的影响



纳秒级调优第三守则：使用专业工具 精准测量

通过 CPU 内部的性能计数器获取其它硬件层面的测量手段提高数据的准确度





你的电脑遇到问题，需要重新启动。
我们只收集某些错误信息，然后你可以重新启动。

30% 完成



有关此问题的详细信息和可能的解决方法，请访问

<https://www.windows.com/stopcode>

如果致电支持人员，请向他们提供以下信息：

终止代码: DRIVER_IRQL_NOT_LESS_OR_EQUAL

失败的操作: sepdrv5.sys

化解之道



USB3.0连接，
5Gb/s

使用独立目标机有很多好处：
OS层的干扰因素少，数据更加一致；
主机不会崩溃，目标机重启很快

试验用目标平台

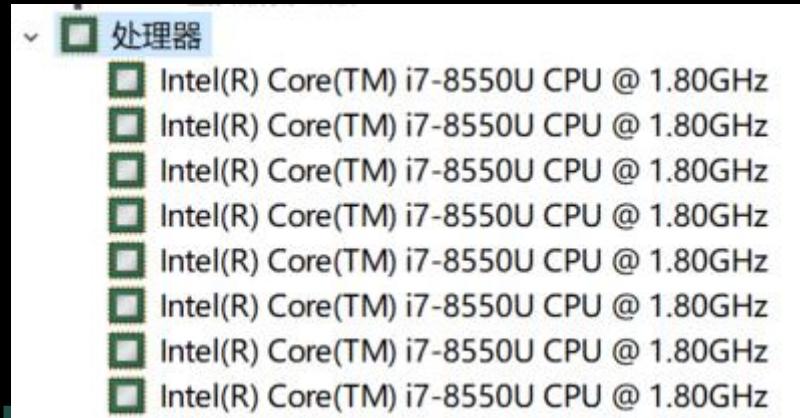


<https://www.nanocode.cn/#/gdk7/index>

- Product Collection
- Intel® Pentium® Processor 4000 Series
- Code Name
- Products formerly Skylake
- Vertical Segment
- Mobile
- Processor Number
- 4405U
- Status
- Launched
- Launch Date
- Q3'15
- Lithography
- 14 nm

试验用主机CPU

- i7 8550U
- P
- O
- D
- I
- C
- U
- S
- M
- L



CPU Specifications

of Cores 4

of Threads 8

Processor Base Frequency 1.80 GHz

Max Turbo Frequency 4.00 GHz

Cache 8 MB Intel® Smart Cache

Bus Speed 4 GT/s

Intel® Turbo Boost Technology 2.0 Frequency 4.00 GHz

TDP 15 W

Configurable TDP-up Frequency 2.00 GHz

Configurable TDP-up 25 W

Configurable TDP-down Frequency 800 MHz

Configurable TDP-down 10 W



如有更多问题，请到异步社区展台单独交流

时间只限今天：2020/12/5

THANKS

Microarchitecture Exploration Microarchitecture Exploration ?

INTEL VTUNE PROFILER

Analysis Configuration Collection Log Summary Bottom-up Event Count Platform gealign.cpp x gealign.cpp x gealign.cpp x

| S... | Source | Assembly | Clockticks | Instructions Retired | CPI Rate | Locators | | | |
|------|---------------------------------|----------|------------|----------------------|----------|----------|-----------------|-----------------|----------------|
| | | | | | | Retiring | Front-End Bound | Bad Speculation | Back-End Bound |
| 165 | } | | | | | | | | |
| 166 | #else | | | | | | | | |
| 167 | for (int i = 0; i < count; i++) | | 1,800,000 | 9,000,000 | 0.200 | 0.0% | 0.0% | 0.0% | 3.8% |
| 168 | { | | | | | | | | |
| 169 | sc[i].c = count * i; | | 27,000,000 | 45,000,000 | 0.600 | 0.0% | 0.0% | 0.0% | 57.7% |
| 170 | } | | | | | | | | |
| 171 | for (int i = 0; i < count; i++) | | 1,800,000 | 7,200,000 | 0.250 | 0.0% | 0.0% | 0.0% | 3.8% |
| 172 | { | | | | | | | | |
| 173 | sum += sc[i].c; | | 16,200,000 | 21,600,000 | 0.750 | 0.0% | 0.0% | 0.0% | 34.6% |
| 174 | } | | | | | | | | |

| Address ▲ | Sour... | Assembly | Clockticks | Instructions Retired | CPI Rate | Locators | | | |
|-------------|---------|---------------------------------|------------|----------------------|----------|----------|-----------------|-----------------|----------------|
| | | | | | | Retiring | Front-End Bound | Bad Speculation | Back-End Bound |
| 0x1400020a5 | 173 | movd xmm2, dword ptr [rax] | 1,800,000 | 3,600,000 | 0.500 | 0.0% | 0.0% | 0.0% | 3.8% |
| 0x1400020a9 | 173 | movd xmm0, dword ptr [rax-0x9] | | | | | | | |
| 0x1400020ae | 173 | movd xmm3, dword ptr [rax-0x12] | 1,800,000 | 1,800,000 | 1.000 | 0.0% | 0.0% | 0.0% | 3.8% |
| 0x1400020b3 | 173 | punpckldq xmm2, xmm1 | | | | | | | |
| 0x1400020b7 | 173 | punpckldq xmm3, xmm0 | | | | | | | |
| 0x1400020bb | 173 | punpcklqdq xmm3, xmm2 | 1,800,000 | 0 | 0.0% | 0.0% | 0.0% | 0.0% | 3.8% |
| 0x1400020bf | 173 | paddd xmm3, xmm5 | 0 | 3,600,000 | 0.000 | 0.0% | 0.0% | 0.0% | 0.0% |
| 0x1400020c3 | 173 | movdqqa xmm5, xmm3 | | | | | | | |
| 0x1400020c7 | 173 | movd xmm1, dword ptr [rax+0x2d] | | | | | | | |
| 0x1400020cc | 173 | movd xmm2, dword ptr [rax+0x24] | 3,600,000 | 3,600,000 | 1.000 | 0.0% | 0.0% | 0.0% | 7.7% |
| 0x1400020d1 | 173 | movd xmm0, dword ptr [rax+0x1b] | | | | | | | |
| 0x1400020d6 | 173 | movd xmm3, dword ptr [rax+0x12] | 1,800,000 | 1,800,000 | 1.000 | 0.0% | 0.0% | 0.0% | 3.8% |
| 0x1400020db | 173 | punpckldq xmm2, xmm1 | 0 | 1,800,000 | 0.000 | 0.0% | 0.0% | 0.0% | 0.0% |
| 0x1400020df | 173 | punpckldq xmm3, xmm0 | 1,800,000 | 0 | 0.0% | 0.0% | 0.0% | 0.0% | 3.8% |
| 0x1400020e3 | 173 | punpcklqdq xmm3, xmm2 | 3,600,000 | 1,800,000 | 2.000 | 0.0% | 0.0% | 0.0% | 7.7% |
| 0x1400020e7 | 173 | paddd xmm3, xmm4 | 0 | 3,600,000 | 0.000 | 0.0% | 0.0% | 0.0% | 0.0% |
| 0x1400020eb | 173 | movdqqa xmm4, xmm3 | | | | | | | |



Microarchitecture Exploration Microarchitecture Exploration

INTEL VTUNE PROFILER

Analysis Configuration Collection Log Summary Bottom-up Event Count Platform gealign.cpp x gealign.cpp x gealign.cpp x gealign.cpp x

Source Assembly



Assembly grouping: Address

| S... | Source | Clockticks | Instructions Retired | CPI Rate | Locators | | | |
|------|---|------------|----------------------|------------|----------|-----------------|-----------------|----------------|
| | | | | | Retiring | Front-End Bound | Bad Speculation | Back-End Bound |
| 166 | #else | | | | | | | |
| 167 | for (int i = 0; i < count; i++) | | 0 | 9,000,000 | 0.000 | 0.0% | 0.0% | 0.0% |
| 168 | { | | | | | | | |
| 169 | sc[i].c = count * i; | | 45,000,000 | 48,600,000 | 0.926 | 41.3% | 0.0% | 41.3% |
| 170 | } | | | | | | | |
| 171 | for (int i = 0; i < count; i++) | | 0 | 5,400,000 | 0.000 | 0.0% | 0.0% | 0.0% |
| 172 | { | | | | | | | |
| 173 | sum += sc[i].c; | 25,200,000 | 23,400,000 | 1.077 | 0.0% | 0.0% | 0.0% | 35.9% |
| 174 | } | | | | | | | |
| 175 | #endif | | | | | | | |
| 176 | uint64_t end = __rdtsc(); | | | | | | | |
| 177 | | | | | | | | |
| 178 | cout << "assign and sum " << name << " took " | | | | | | | |

| Address ▲ | Sour... | Assembly | Clockticks | Instructions Retired | CPI Rate | Locators | | | |
|-------------|---------|---------------------------------|------------|----------------------|----------|----------|-----------------|-----------------|----------------|
| | | | | | | Retiring | Front-End Bound | Bad Speculation | Back-End Bound |
| 0x1400023e5 | 173 | movd xmm2, dword ptr [rax] | 5,400,000 | 5,400,000 | 1.000 | 0.0% | 0.0% | 0.0% | 7.7% |
| 0x1400023e9 | 173 | movd xmm0, dword ptr [rax-0x10] | 0 | 1,800,000 | 0.000 | 0.0% | 0.0% | 0.0% | 0.0% |
| 0x1400023ee | 173 | movd xmm3, dword ptr [rax-0x20] | 3,600,000 | 0 | 0.000 | 0.0% | 0.0% | 0.0% | 5.1% |
| 0x1400023f3 | 173 | punpckldq xmm2, xmm1 | | | | | | | |
| 0x1400023f7 | 173 | punpckldq xmm3, xmm0 | 1,800,000 | 1,800,000 | 1.000 | 0.0% | 0.0% | 0.0% | 2.6% |
| 0x1400023fb | 173 | punpcklqdq xmm3, xmm2 | | | | | | | |
| 0x1400023ff | 173 | paddd xmm3, xmm5 | 0 | 1,800,000 | 0.000 | 0.0% | 0.0% | 0.0% | 0.0% |
| 0x140002403 | 173 | movdqqa xmm5, xmm3 | | | | | | | |
| 0x140002407 | 173 | movd xmm1, dword ptr [rax+0x50] | 1,800,000 | 0 | 0.000 | 0.0% | 0.0% | 0.0% | 2.6% |
| 0x14000240c | 173 | movd xmm2, dword ptr [rax+0x40] | 3,600,000 | 5,400,000 | 0.667 | 0.0% | 0.0% | 0.0% | 5.1% |
| 0x140002411 | 173 | movd xmm0, dword ptr [rax+0x30] | | | | | | | |
| 0x140002416 | 173 | movd xmm3, dword ptr [rax+0x20] | 1,800,000 | 0 | 0.000 | 0.0% | 0.0% | 0.0% | 2.6% |
| 0x14000241b | 173 | punpckldq xmm2, xmm1 | | | | | | | |

/arch:IA32

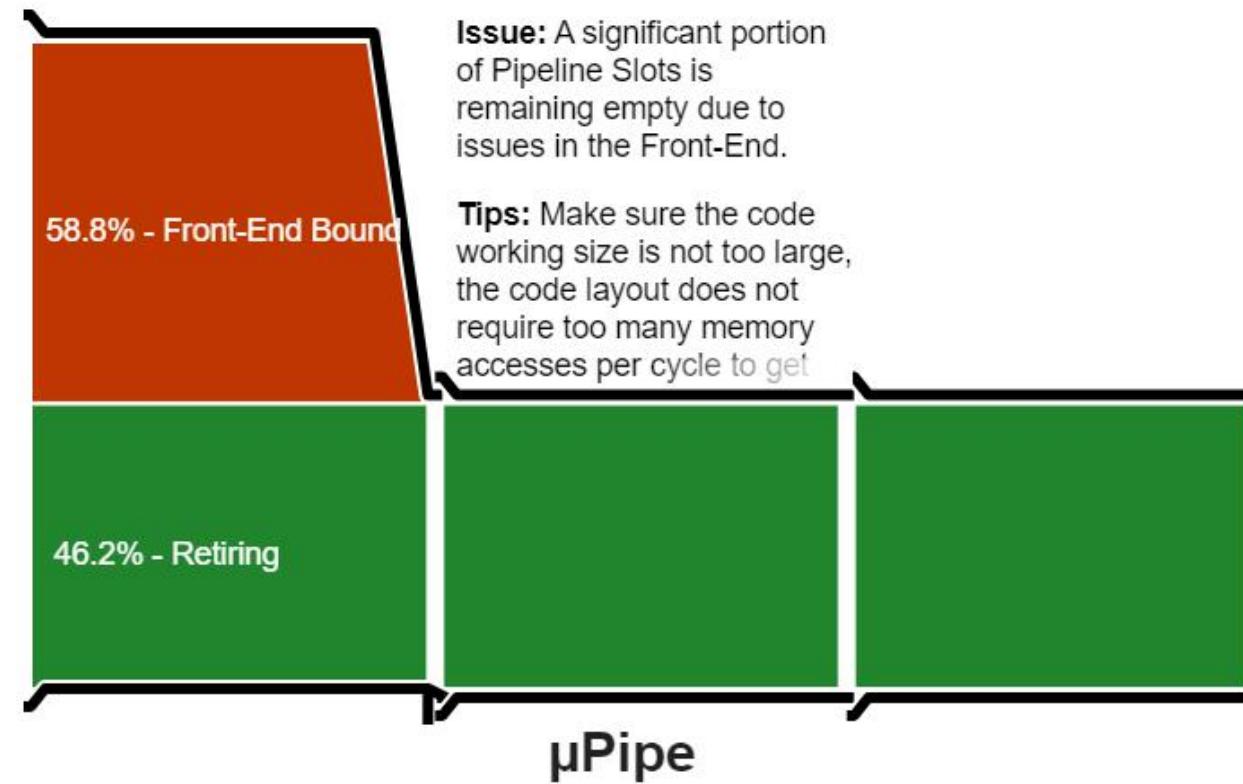
```
multiply aligned n with i 100000 times= 704982704 took 479734 ticks
multiply unaligned n with i 100000 times= 704982704 took 483548 ticks
```

```
multiply aligned n with i 100000 times= 704982704 took 555849 ticks
multiply unaligned n with i 100000 times= 704982704 took 610758 ticks
```

```
multiply aligned n with i 100000 times= 704982704 took 472662 ticks
multiply unaligned n with i 100000 times= 704982704 took 525922 ticks
```

Elapsed Time [?]: 59.534s

| | |
|---|---------------------------|
| Clockticks: | 1,947,600,000 |
| Instructions Retired: | 691,200,000 |
| CPI Rate [?] : | 2.818 ↘ |
| MUX Reliability [?] : | 0.933 |
| Retiring [?] : | 46.2% ↘ of Pipeline Slots |
| General Retirement [?] : | 14.3% of Pipeline Slots |
| Microcode Sequencer [?] : | 31.9% ↘ of Pipeline Slots |
| Front-End Bound [?] : | 58.8% ↘ of Pipeline Slots |
| Front-End Latency [?] : | 45.8% ↘ of Pipeline Slots |
| Front-End Bandwidth [?] : | 13.0% ↘ of Pipeline Slots |
| Front-End Bandwidth MITE [?] : | 0.0% of Clockticks |
| Front-End Bandwidth DSB [?] : | 9.4% of Clockticks |
| (Info) DSB Coverage [?] : | 28.1% ↘ |
| Bad Speculation [?] : | 0.0% of Pipeline Slots |
| Back-End Bound [?] : | 0.0% of Pipeline Slots |
| Average CPU Frequency [?] : | 1.8 GHz |
| Total Thread Count: | 2 |
| Paused Time [?] : | 0s |



This diagram represents inefficiencies in CPU usage. Treat it as a pipe with an output flow equal to the "pipe efficiency" ratio: (Actual Instructions Retired)/(Maximum Possible Instruction Retired). If there are pipeline stalls decreasing the pipe efficiency, the pipe shape gets more narrow.