

(not really exception-specific)

Part of function specification

- Never Fails
- Strong Exception Guarantee:
 - may fail (throw), but will restore program state to what it was before: transactional
 - possible and desirable in library functions
 - very hard in application code
 - usually too many state changes

(not really exception-specific)

Part of function specification

- Never Fails
- Strong Exception Guarantee:
 - may fail (throw), but will restore program state to what it was before: transactional
 - possible and desirable in library functions
 - very hard in application code
 - usually too many state changes
- Basic Exception Guarantee:
 - may fail (throw), but will restore program to some valid state

Basic Exception Safety Guarantee



Customer: "Hello, is this Microsoft Word support? I was writing a book. Suddenly, Word deleted everything."

Microsoft: "Oh, that's ok. Word only provides a basic exception guarantee."

Customer: "Oh, alright then, thank you very much and have a good day!"

- Error handling is a lot of effort
 - in development
 - must be paranoid
 - create a lot of extra code
 - in testing
 - many codepaths to test
 - if you don't test them, they won't work

- Error handling is a lot of effort
 - in development
 - must be paranoid
 - create a lot of extra code
 - in testing
 - many codepaths to test
 - if you don't test them, they won't work
- Little customer gain

- Error handling is a lot of effort
 - in development
 - must be paranoid
 - create a lot of extra code
 - in testing
 - many codepaths to test
 - if you don't test them, they won't work
- Little customer gain
- So what do we do?

So what do we do?

- Check everything
 - check every API call
 - one wrapper per error reporting method
 - Windows: `GetLastError()`, `HRESULT`
 - Unix: `errno`
 - `assert` aggressively
 - asserts stay in Release
 - `noexcept` if caller does not handle exception
 - `std::terminate`, but unexpected exceptions will terminate anyway
 - install handler with `std::set_terminate` for checking

So what do we do?

- Check everything
 - check every API call
 - one wrapper per error reporting method
 - Windows: `GetLastError()`, `HRESULT`
 - Unix: `errno`
 - `assert` aggressively
 - asserts stay in Release
 - `noexcept` if caller does not handle exception
 - `std::terminate`, but unexpected exceptions will terminate anyway
 - install handler with `std::set_terminate` for checking
- Assume everything works

So what do we do?

- Check everything
 - check every API call
 - one wrapper per error reporting method
 - Windows: `GetLastError()`, `HRESULT`
 - Unix: `errno`
 - `assert` aggressively
 - asserts stay in Release
 - `noexcept` if caller does not handle exception
 - `std::terminate`, but unexpected exceptions will terminate anyway
 - install handler with `std::set_terminate` for checking
- Assume everything works
- Goal:
 - keep set of code paths small
 - keep set of program states small

- prio 1: collect as much information as possible
 - client: send core dump home
 - server: halt thread and notify operator

- prio 1: collect as much information as possible
 - client: send core dump home
 - server: halt thread and notify operator
- prio 2: carry on somehow
 - if check was critical, program behavior now undefined: no further reports
 - never terminate!
 - asserts can be wrong, too!
 - if you need safety (nuclear powerplant, etc.), add at higher level
 - example: server stops processing request categories with too many pending requests

- Reproduce the error in the lab
- Add handling code only for errors that are reproducible
 - Otherwise you write
 - error handlers that are never used
 - error handlers that are never tested, do the wrong thing

5% of handlers handle 95% of errors

- Reproduce the error in the lab
- Add handling code only for errors that are reproducible
 - Otherwise you write
 - error handlers that are never used
 - error handlers that are never tested, do the wrong thing

5% of handlers handle 95% of errors

- Not all errors equal

- `nullptr` access
- API calls not expected to fail (in this way)
- assertions

- `nullptr` access
- API calls not expected to fail (in this way)
- assertions
- "never happens"
 - no handler
 - like C++ undefined behavior: program is invalid

- `nullptr` access
- API calls not expected to fail (in this way)
- assertions
 - "never happens"
 - no handler
 - like C++ undefined behavior: program is invalid
- Client: send report, disable future reports
- Server: send report, infinite loop (wait for debugger)
- Notify user only if false alarm unlikely
 - assertions may be wrong

```
auto RegisterFooHook(Foo foo) {
    errcode_t err=RegisterFoo(foo);
    if(err==SUCCESS) KeepTrackOfFoo(foo);
    return err;
}
```

- If `err` indicates error, do nothing

```
auto RegisterFooHook(Foo foo) {
    errcode_t err=RegisterFoo(foo);
    if(err==SUCCESS) KeepTrackOfFoo(foo);
    return err;
}
```

- If `err` indicates error, do nothing
- But no reproduction for `RegisterFoo` failing
- Effect on rest of the program?

```
auto RegisterFooHook(Foo foo) {
    errcode_t err=RegisterFoo(foo);
    if(err==SUCCESS) KeepTrackOfFoo(foo);
    return err;
}
```

- If `err` indicates error, do nothing
- But no reproduction for `RegisterFoo` failing
- Effect on rest of the program?
- Client: send report, throttle future similar reports
 - in Debug: notify developer
- Server: send report

- Reproducible 3rd party bug
 - sometimes PowerPoint makes shape disappear
- Reproducible, tested and supported

- Reproducible 3rd party bug
 - sometimes PowerPoint makes shape disappear
- Reproducible, tested and supported
- Not nice, users may complain

- Reproducible 3rd party bug
 - sometimes PowerPoint makes shape disappear
- Reproducible, tested and supported
- Not nice, users may complain
- Client/Server: only log, no report
 - to explain behavior if user calls

Indication of broken environment

- Other add-in hooked same function as us
- OS reports space as default decimal separator
 - both fully supported by us

- Other add-in hooked same function as us
- OS reports space as default decimal separator
 - both fully supported by us
- Could still be cause of a problem

- Other add-in hooked same function as us
- OS reports space as default decimal separator
 - both fully supported by us
- Could still be cause of a problem
- Client during remote support: notify support engineer
 - maybe reason for support call

- Reports with core dumps sent to server
 - automatically
 - if user opted out, user can send prepared email

- Reports with core dumps sent to server
 - automatically
 - if user opted out, user can send prepared email
- Error database
 - core dumps opened in debugger
 - errors automatically categorized by file/line
 - details and core dump accessible to devs

- Reports with core dumps sent to server
 - automatically
 - if user opted out, user can send prepared email
- Error database
 - core dumps opened in debugger
 - errors automatically categorized by file/line
 - details and core dump accessible to devs
- Devs can mark errors as fixed
 - trigger automatic update
 - or send automatic email - magic!

- Problem often related to customer environment
 - Proxy: list of loaded modules (DLLs, dylibs) in dump
- Can we identify module causing error?
 - or versions of module?

- Problem often related to customer environment
 - Proxy: list of loaded modules (DLLs, dylibs) in dump
- Can we identify module causing error?
 - or versions of module?

Report database with all reports

- 1 means has particular problem
- 0 means has different problem

`0 1 1 0 0 1 0 1 0 1 1 0` (6 occurrences among 12 reports)

- Problem often related to customer environment
 - Proxy: list of loaded modules (DLLs, dylibs) in dump
- Can we identify module causing error?
 - or versions of module?

Report database with all reports

- 1 means has particular problem
- 0 means has different problem

`0 1 1 0 0 1 0 1 0 1 1 0` (6 occurrences among 12 reports)

`x - x - - x x - x - x -` Module A (with: 3/6, without: 3/6)

`- x x - x x - x x - - -` Module B (with: 4/6, without: 2/6)

- Problem often related to customer environment
 - Proxy: list of loaded modules (DLLs, dylibs) in dump
- Can we identify module causing error?
 - or versions of module?

Report database with all reports

- 1 means has particular problem
- 0 means has different problem

`0 1 1 0 0 1 0 1 0 1 1 0` (6 occurrences among 12 reports)

`x - x - - x x - x - x -` Module A (with: 3/6, without: 3/6)

`- x x - x x - x x - - -` Module B (with: 4/6, without: 2/6)

- Module B responsible? Or chance?

- Compressing

`0 1 1 0 0 1 0 1 0 1 1 0` (6/12)

- Knowing if reports contain module B helps compressing?

`- x x - x x - x x - - -` Module B (with: 4/6, without: 2/6)

- Compressing

0 1 1 0 0 1 0 1 0 1 1 0 (6/12)

- Knowing if reports contain module B helps compressing?

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

- perfect arithmetic compression (Laplacian estimator)
 - estimates probability p that report has particular problem
- all p elem [0, 1] equally likely
- no. bits to compress N bits with K ones:

$\log [(N+1) * (N \text{ over } K)]$

- no. bits becomes smaller if p is closer to 0 or 1:
 - 12 bits with 6 ones: 13.55 bits
 - 12 bits with no ones: 3.70 bits

Compressing Reports

0 0 1 0 0 1 0 1 0 1 1 0 (6/12)

x - x - - x x - x - x - Module A (with: 3/6, without: 3/6)

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

Compressing Reports

0 0 1 0 0 1 0 1 0 1 1 0 (6/12)

x - x - - x x - x - x - Module A (with: 3/6, without: 3/6)

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

- Compressing all reports together (6/12): 13.55 bits

0 0 1 0 0 1 0 1 0 1 1 0 (6/12)

x - x - - x x - x - x - Module A (with: 3/6, without: 3/6)

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

- Compressing all reports together (6/12): 13.55 bits
- Make use of module A
 - choose module A over B: 1 bit
 - compressing all reports with A (3/6): 7.13 bits
 - compressing all reports without A (3/6): 7.13 bits
 - total: 15.26 bits - module A has nothing to do with problem

0 0 1 0 0 1 0 1 0 1 1 0 (6/12)

x - x - - x x - x - x - Module A (with: 3/6, without: 3/6)

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

- Compressing all reports together (6/12): 13.55 bits
- Make use of module B
 - choose module B over A: 1 bit
 - compressing all reports with B (4/6): 6.71 bits
 - compressing all reports without B (2/6): 6.71 bits
 - total: 14.43 bits - still not relevant enough

0 0 1 0 0 1 0 1 0 1 1 0 (6/12)

x - x - - x x - x - x - Module A (with: 3/6, without: 3/6)

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

- Compressing all reports together (6/12): 13.55 bits
- Make use of module C
 - choose module C over A and B: 1.58 bits
 - compressing all reports with C (5/6): 5.39 bits
 - compressing all reports without C (1/6): 5.39 bits
 - total: 12.37 bits - relevant!

0 0 1 0 0 1 0 1 0 1 1 0 (6/12)

x - x - - x x - x - x - Module A (with: 3/6, without: 3/6)

- x x - x x - x x - - - Module B (with: 4/6, without: 2/6)

- Compressing all reports together (6/12): 13.55 bits
- Make use of module C
 - choose module C over A and B: 1.58 bits
 - compressing all reports with C (5/6): 5.39 bits
 - compressing all reports without C (1/6): 5.39 bits
 - total: 12.37 bits - relevant!
- Extend to module versions
- More hypotheses make chance more likely

- new language feature
- `assert` on steroids
- declarative function pre- and postconditions

```
void push(int x, queue& q)
[[expects: !q.full()]]
[[ensures: !q.empty()]]
{
...
[[assert: q.is_valid()]]
...
}
```

- When check contract?
 - debug
 - release
 - never
- What to do if contract violated?
 - terminate
 - carry on
 - report (what to whom?)

- When check contract?
 - debug
 - release
 - never
- What to do if contract violated?
 - terminate
 - carry on
 - report (what to whom?)
- removed from C++20 at last moment
- discussion will continue for C++23

THANK YOU!



for attending.

And yes, we are recruiting:

CPP-Summit

A Very Special Class of Errors

```
std::int32_t a=2 000 000 000;  
std::int32_t b=a+a;
```

What is **b**?

```
std::int32_t a=2 000 000 000;  
std::int32_t b=a+a;
```

What is **b**?

Uuh, may overflow.

- Let's check for it!

```
if( b<a ) {  
... treat overflow ...  
}
```

Ok?

Undefined Behavior (UB)

Example: int arithmetic overflow

```
std::int32_t a=2 000 000 000;  
std::int32_t b=a+a;
```

What is **b**?

Uuh, may overflow.

- Let's check for it!

```
if( b<a ) {  
... treat overflow ...  
}
```

Ok?

Undefined Behavior (UB)

Example: int arithmetic overflow

If program contains undefined behavior, compiler can do anything *with the whole program!*

- In particular, compiler may assume that UB never happens



Charley Bay

系统架构专家

Charley 是知名的系统架构专家，他拥有超过三十年的软件开发经验，使用C++在多个控制和高性能的领域，特别集中在大规模和分布式系统等性能敏感的环境，包括大数据集的时间敏感处理，性能可视化，实时处理，低延迟，嵌入式，以及系统状态和控制。

主办方：

Boolan
高端IT咨询与教育平台

System Architecture And Design

Contrasting { Development Space,
Architectural Space,
Design Space.

CPP-Summit 2020

C++ and System Software Summit

📅 December 4-5

📍 Hyatt Regency Shenzhen Airport

charley bay

“Space”

Space (def): The gradient of concerns that defines possibilities

Space dictates:

- What concepts are represented
- What issues and solutions are possible

- Is often N-dimensional
- Each concern may be weighted
- Each concern may be ranked



Space implies:

- Some issues and solutions are simply discussed (*the space represents the domain for exploring those issues and solutions*)
- Some issues or solutions are invisible (*the space does not represent them*):
 1. Issue is orthogonal to that space (*unrelated*)
 2. Issue cannot be described nor addressed (*must be managed by a different space*)

Think In Terms Of...

As Developers,

We think in terms of:

- Programming Language (*syntactic and semantic rules*)
- Paradigm (*“How To Think” in breaking down a problem*)
- The Problem To Be Addressed (*what technical and ergonomic concerns exist?*)

Think In Terms Of...

As Developers,

We think in terms of:

- Programming Language (*syntactic and semantic rules*)
- Paradigm (*“How To Think” in breaking down a problem*)
- The Problem To Be Addressed (*what technical and ergonomic concerns exist?*)

Is all about:
**Technical
Solutions**

Think In Terms Of...

As Developers,

We think in terms of:

- Programming Language (*syntactic and semantic rules*)
- Paradigm (*"How To Think" in breaking down a problem*)
- The Problem To Be Addressed (*what technical and ergonomic concerns exist?*)

Is all about:
**Technical
Solutions**

As Experienced Developers,

We also think about:

- The Development Process (*Traditional and Iterative domain exploration*)
- Roles played (*interaction by diverse interested parties*)
- Dimensions defining the “spaces” in which:
 - the problem is defined
 - the solution is expressed

Think In Terms Of...

As Developers,

- Programming Language (*syntactic and semantic rules*)
- Paradigm (*"How To Think" in breaking down a problem*)
- The Problem To Be Addressed (*what technical and ergonomic concerns exist?*)

As Experienced Developers,

- The Development Process (*Traditional and Iterative domain exploration*)
- Roles played (*interaction by diverse interested parties*)
- Dimensions defining the “spaces” in which:
 - the problem is defined
 - the solution is expressed

Is all about:
Technical Solutions

We also think about:

Is all about:
**Business Coordination,
Constraints Management**

“Role”

Role (*def*): A function or part performed

Example roles:

Developer, Designer, Architect,
Domain Expert, Product Owner, Program Manager,
Customer Advocate, Executive Owner

Role dictates:

- What you care about
- What you are responsible for

Role implies:

- Your concerns are unique (*other roles have different concerns*)
- You must interact with other roles (*others are worried about different but overlapping concerns*)

Roles Change Over Time

New Developers tend to focus on:

- How does this C++ language feature work?
- How to implement a technical solution?
- What is Best Practice?

*Is plenty
to explore!
(technical skills
must be acquired)*

Roles Change Over Time

New Developers tend to focus on:

- How does this C++ language feature work?
- How to implement a technical solution?
- What is Best Practice?

*Is plenty
to explore!
(technical skills
must be acquired)*

Designers tend to focus on:

- What design approach should this system prefer?
- What paradigms (*how to think*) should be used?
- How should our code change: What C++ Language additions enable new idioms to solve past problems?

*Favor addressing some concerns,
over a different design that addresses
differently ranked concerns*

Roles Change Over Time

New Developers tend to focus on:

- How does this C++ language feature work?
- How to implement a technical solution?
- What is Best Practice?

Both care about
C++ Language Features,
but for different reasons!

*Is plenty
to explore!
(technical skills
must be acquired)*

Designers tend to focus on:

- What design approach should this system prefer?
- What paradigms (*how to think*) should be used?
- How should our code change: What C++ Language additions enable new idioms to solve past problems?

*Favor addressing some concerns,
over a different design that addresses
differently ranked concerns*

Roles Change Over Time *(continued)*

Architects tend to focus on:

- What should the **system do?**
- What **technologies** should we leverage
(to permit the system to be implemented and function as expected)?
- What **reuse** should we manage?
- What organizational **technical evolution** should we pursue?

Roles Change Over Time *(continued)*

Architects tend to focus on:

- What should the **system do?**
- What **technologies** should we leverage (*to permit the system to be implemented and function as expected*)?
- What **reuse** should we manage?
- What organizational **technical evolution** should we pursue?

Primary considerations:

1. Solution Space
2. Product Space
3. Organizational Health

Roles Change Over Time *(continued)*

Architects tend to focus on:

- What should the **system do?**
- What **technologies** should we leverage (*to permit the system to be implemented and function as expected*)?
- What **reuse** should we manage?
- What organizational **technical evolution** should we pursue?

Primary considerations:

1. Solution Space
2. Product Space
3. Organizational Health

Secondary considerations:

1. External technological landscape (*including C++ Language advances*)

Caring About C++ Language Evolution

Developers in all roles care about C++ Language Features,
but for different reasons!

New Developer

1. How do I do my job?
2. What is Best Practice?

Caring About C++ Language Evolution

Developers in all roles care about C++ Language Features,
but for **different reasons!**

New Developer

1. How do I **do my job?**
2. What is **Best Practice?**

Experienced Designer

1. What **design idioms** are now available?
2. What problems can I solve now, in a more **robust** or **elegant** manner?

Caring About C++ Language Evolution

Developers in all roles care about C++ Language Features,
but for different reasons!

New Developer

1. How do I do my job?
2. What is Best Practice?

Experienced Designer

1. What design idioms are now available?
2. What problems can I solve now, in a more robust or elegant manner?

Architect

1. How should our products change (*use of hardware and software*)?
2. What C++ Language features make it compelling...
 - To prefer C++ over other languages
 - To prefer C++ over alternative (*such as implementing subsystem in hardware*)

Developer's Perspective Of Roles

- Q: How is “Experienced Designer” different from “Architect”?

Developer's Perspective Of Roles

- Q: How is “Experienced Designer” different from “Architect”?
- A:

Architect

is concerned about the
business direction

Developer's Perspective Of Roles

- Q: How is “Experienced Designer” different from “Architect”?
- A:

Architect

is concerned about the business direction

Designer

is concerned about the successful technical delivery

Developer's Perspective Of Roles

- Q: How is “Experienced Designer” different from “Architect”?
- A:

Architect

is concerned about the business direction

Designer

is concerned about the successful technical delivery

New Developer

is concerned about making sense of all the “sparkly” things
(not yet able to rank importance among topics)

How Development Works

Non-Optional: Architecture and Design

Review: Development Overview

Process Artifacts

(things delivered)

- The Problem Space defines the Business Need
...leading to
 - System Analysis defining “What needs to be addressed”
...leading to
 - Proposed Architectures defining “How to establish a solution”

Review: Development Overview

- The Problem Space defines the Business Need
...leading to
- System Analysis defining “What needs to be addressed”
...leading to
- Proposed Architectures defining “How to establish a solution”

Process Artifacts

(things delivered)

Business Leadership

Prioritize within the problem space

Technical Leadership

(Architects and Designers) manage technical aspects that define the solution space

Roles and responsibilities are unique and not overlapping; but in practice are often confused due to tradeoffs resulting from:

- Constraints
- Preferences
- Capabilities
- Cross-functional ranking of requirements

Roles Played

(responsible actors)

The “Olden Days”

1

Requirements Analysis

- Specification for what the system must do
- Who does this? → Business Managers

2

Architecture & Design

- Proposed system to deliver that specified in 1
- Who does this? → Architects & Designers

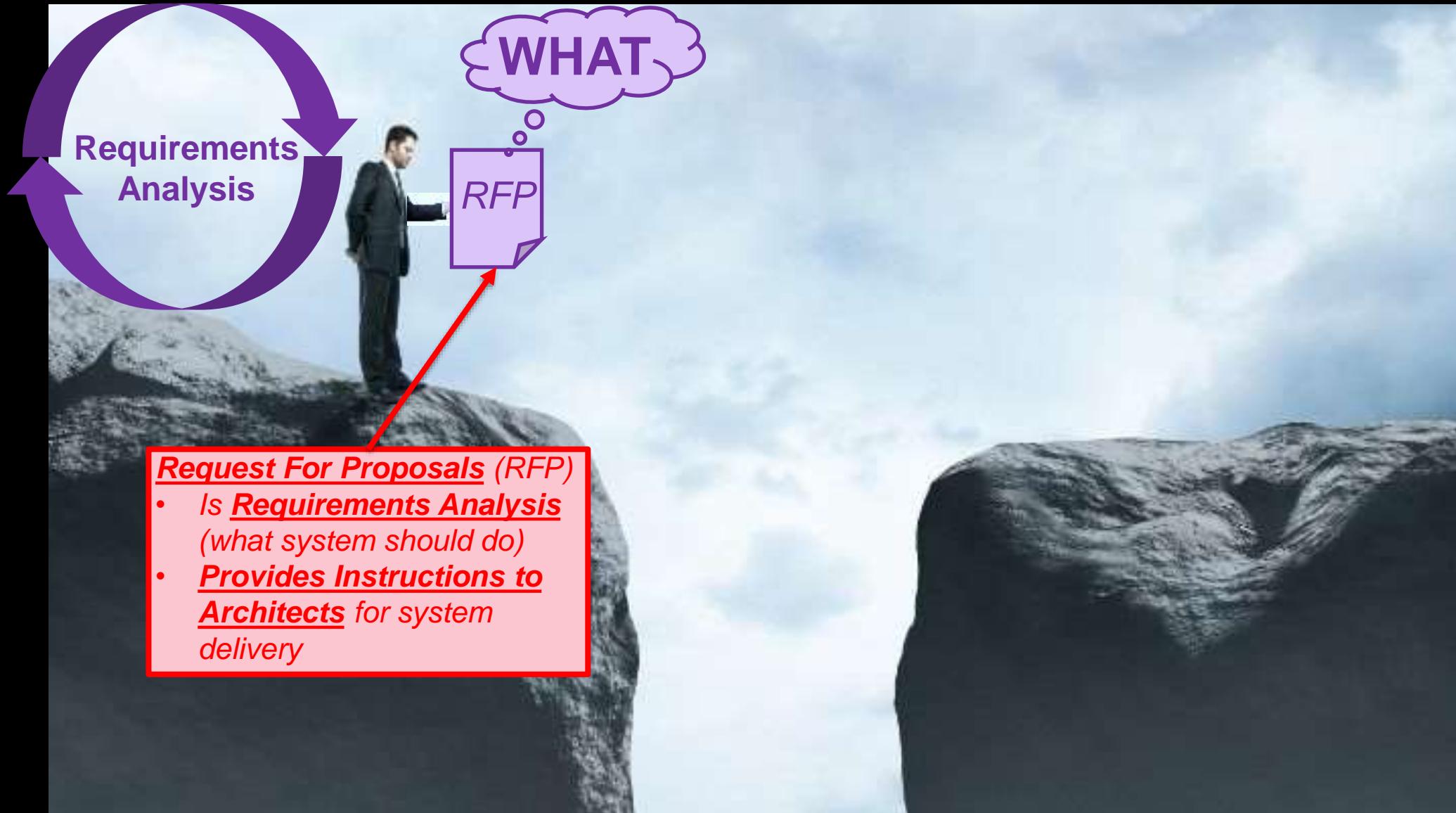
3

Implementation

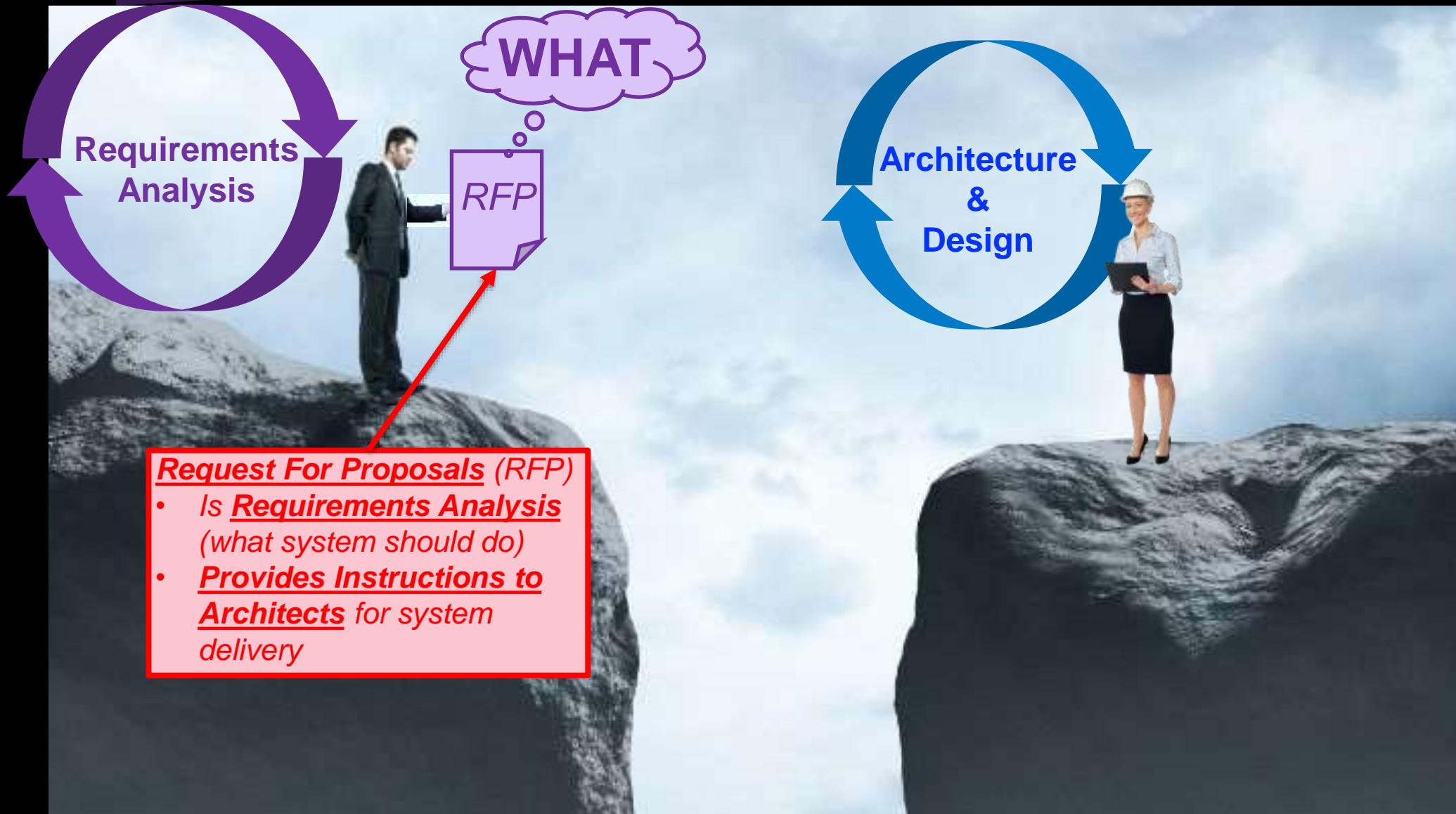
- Construct system specified in 2
- Who does this? → Architects & Designers & Developers



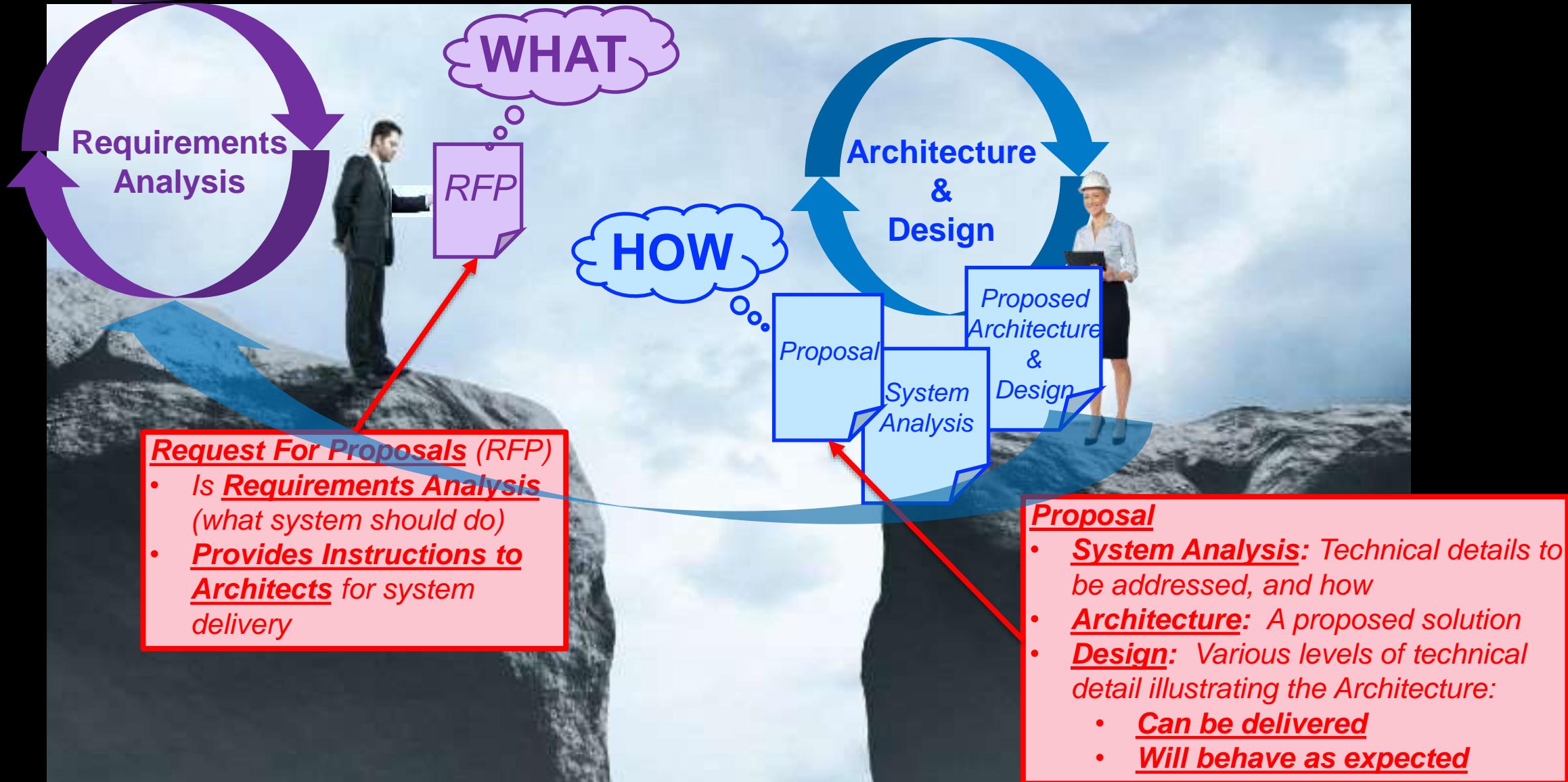
The “Olden Days” (*same thing, different view*)



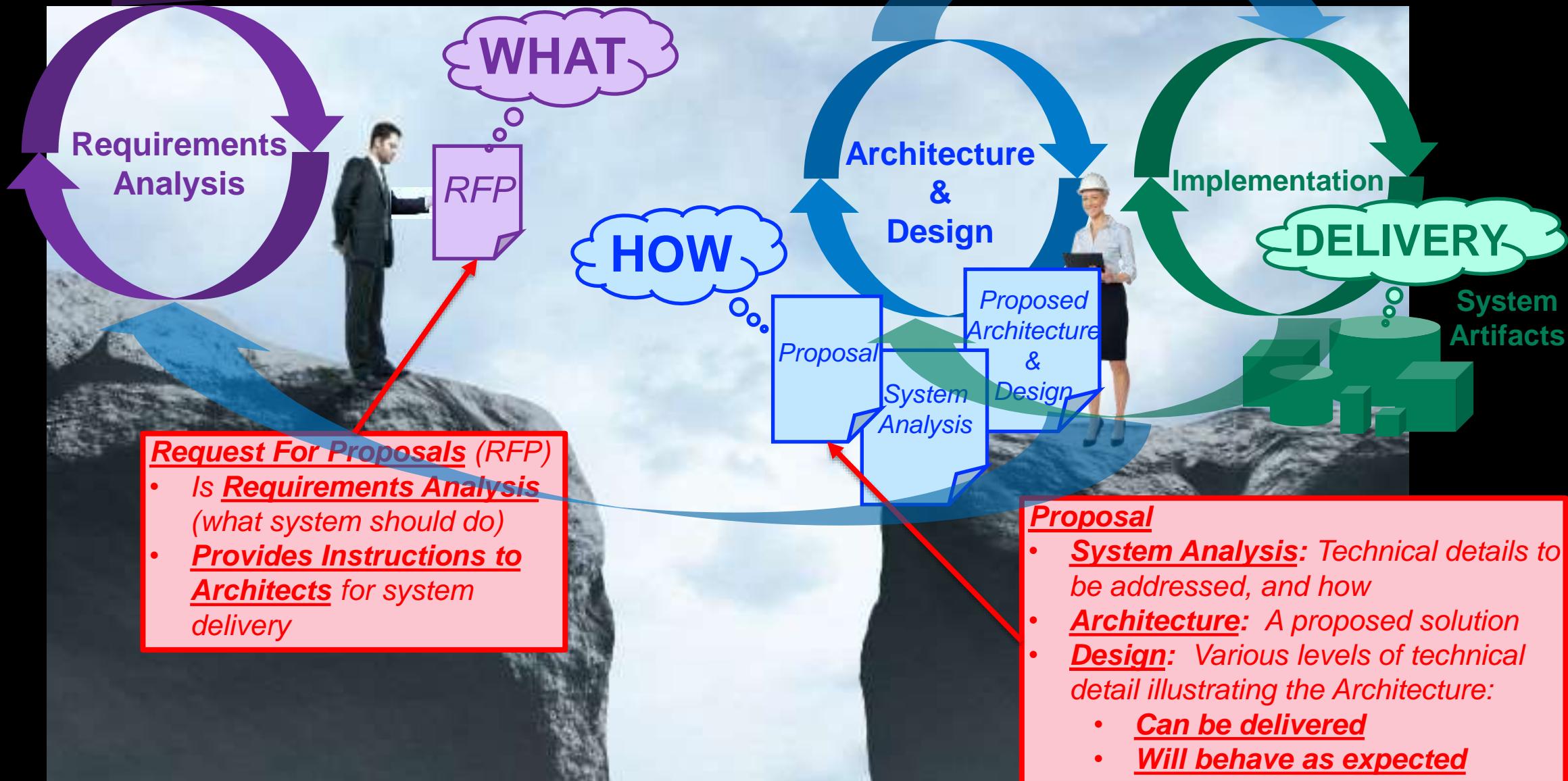
The “Olden Days” (*same thing, different view*)



The “Olden Days” (same thing, different view)



The “Olden Days” (same thing, different view)



Should Not Mix

- Some things you should not mix:
 - Do not mix beer with wine
 - Do not mix business with pleasure
 - Do not mix ammonia with bleach
 - Do not mix ketchup with baking soda

Bad Things Happen™

When you mix what should not be mixed



when you mix ketchup and baking soda.

Ketchup and Baking Soda Prank (GONE WRONG) At Home

1.9M views

2.6K

206

SHARE

SAVE

...

<https://www.youtube.com/watch?v=fov9cj5Vsj4>

Never Mix

- Some things you can **NEVER MIX**
 - You must **ALWAYS know** when you are doing “one” or the “other”
 - If you discover you are **unsure which** you are doing, **STOP** (*and figure it out*)

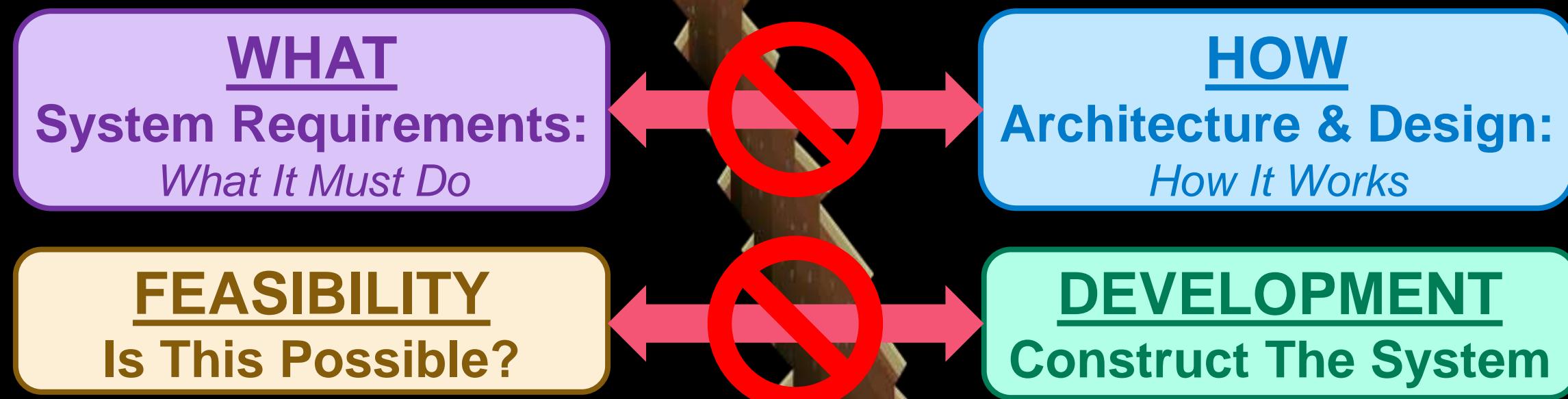
Never Mix

- Some things you can **NEVER MIX**
 - You must **ALWAYS know** when you are doing “one” or the “other”
 - If you discover you are **unsure which** you are doing, **STOP** (*and figure it out*)
- NEVER MIX:



Never Mix

- Some things you can **NEVER MIX**
 - You must **ALWAYS know** when you are doing “one” or the “other”
 - If you discover you are **unsure which** you are doing, **STOP** (*and figure it out*)
- NEVER MIX:



System Analysis

Given: System Requirements

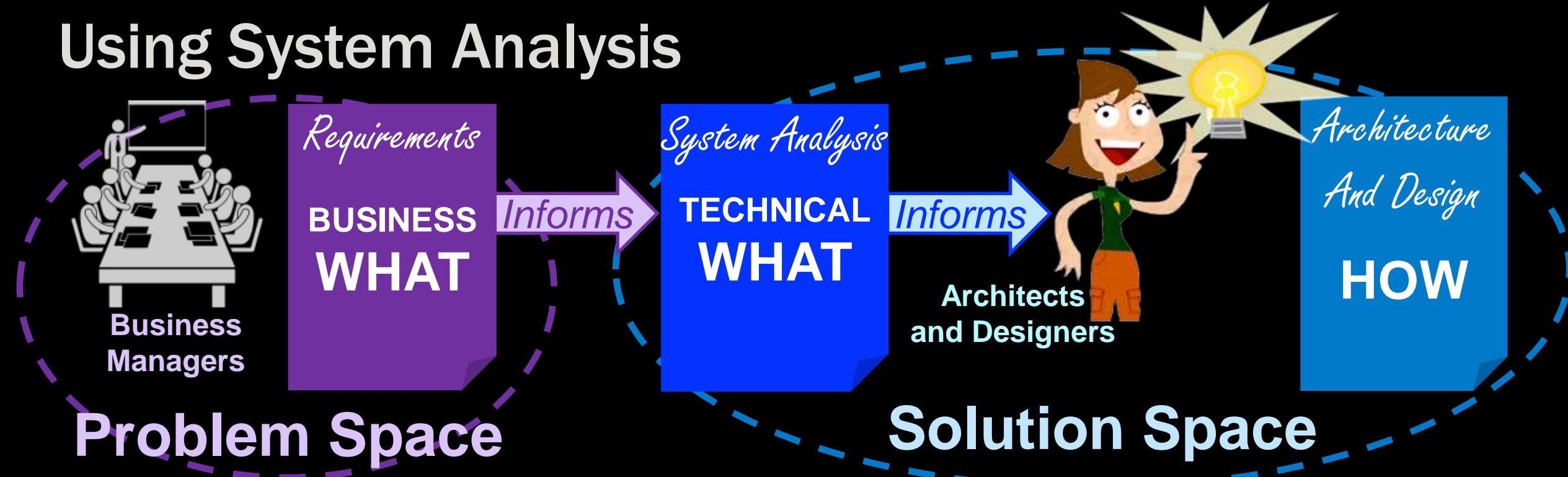
Perform: Technical breakdown of essential system behavior

- **Latency**: How fast must the system respond?
 - What things must be low-latency (*quantify it!*)
 - What acceptable latency tolerances (*by event-category and processing-category*)?
- **Throughput**: What aggregate throughput is required?
 - “Big-pipes”: Data acquisition, stream processing, client-transfer
 - “Small-pipes”: Status-and-control, health monitoring, telemetry
- **Sensitivity**: What “lost events” are tolerable?
 - Can we lose some status-and-control, if we log the failure?
 - Can we miss some acquisition events (*without detecting the event?*)

Common Examples

(must use criteria for your system)

Using System Analysis



After Requirements, and System Analysis:

- Conceptualize scenarios and frameworks that achieve what is required from system analysis
 - Each conception is an architectural option (*defines the total system Theory of Operation*)
 - Each conception for a subsystem is a design option (*defines subsystem Theory of Operation, which must align within the whole system conception*)
- Enables delivery of system with required behavior

Be Wary: Overspecification

Overspecification (def):
An erroneous constraint

Why Spiral / Agile?

Erroneous constraints always provide negative value

Because overspecification hurts your system (and you!)

We are often tempted to “over-specify”:

1. Specify (prescribe) detail which is not necessary detail
2. Assume necessary behavior (*which is not necessary behavior*)

Spiral / Agile tends to “skip” (by “jumping over”) this part of the process, to not be caught by these mistakes

Overspecifications are identified by (continually) re-visiting “First Principles”
(i.e., the system’s Purpose)

Corollary: Any assumption unrelated to purpose is not a “real” specification/constraint

One might surmise that the entire Agile movement is motivated solely through a fear of overspecification

A Real Problem™
That Agile
does Really Solve™

Overspecification Hurts

Overspecification (def):
An erroneous constraint

Erroneous constraints always provide negative value

Over-Specifications

- Limit options
 - For no benefit
 - For negative benefit
- Confuse/Hide the actual constraints!
 - Harder to discover elegant patterns
 - Harder to reason about (*understand*) behavior
- Slow our systems
 - by adding unnecessary constraints
- Break our systems
 - when unnecessary constraints are (*eventually*) violated

Over-specifications
are identified by (continually)
re-visiting “First Principles”
(i.e., the system’s Purpose)



Corollary: Any assumption
unrelated to purpose is not a
“real” specification/constraint

Review: Terms To Limit Your System

- These terms relate to “limits” in delivery of Your System:

Requirement (*def*): A mandated specification

Constraint (*def*): A limit of possibility

Preference (*def*): An expressed bias to rank alternatives

Capability (*def*): An ability to deliver

Understand Your (System-)Limits

Overspecification is a painful limit, but only one limit.

As architect, you balance many limits:

Requirement

You are limited by mandate (*specification*) for what system must do

Understand Your (System-)Limits

Overspecification is a painful limit, but only one limit.

As architect, you balance many limits:

Requirement

You are limited by mandate (*specification*) for what system must do

Constraint

You are limited by what is possible (*limits of technology or physics*)

Understand Your (System-)Limits

Overspecification is a painful limit, but only one limit.

As architect, you balance many limits:

Requirement

You are limited by mandate (*specification*) for what system must do

Constraint

You are limited by what is possible (*limits of technology or physics*)

Preference

You are limited by bias expressing preferred tradeoffs

Understand Your (System-)Limits

Overspecification is a painful limit, but only one limit.

As architect, you balance many limits:

Requirement

You are limited by mandate (*specification*) for what system must do

Constraint

You are limited by what is possible (*limits of technology or physics*)

Preference

You are limited by bias expressing preferred tradeoffs

Capability

You are limited by the implementation team (*what they understand, and can deliver; their availability and capabilities*)

Understand Your (System-)Limits

Overspecification is a painful limit, but only one limit.

As architect, you balance many limits:

Requirement

You are limited by mandate (*specification*) for what system must do

How to relax?

Constraint

You are limited by what is possible (*limits of technology or physics*)

How to relax?

Preference

You are limited by bias expressing preferred tradeoffs

How to relax?

Capability

You are limited by the implementation team (*what they understand, and can deliver; their availability and capabilities*)

How to relax?

Understand Your (System-)Limits

Overspecification is a painful limit, but only one limit.

As architect, you balance many limits:

Requirement

You are limited by mandate (*specification*) for what system must do

How to relax?

- Can: Negotiate the mandate, or change “Acceptance Criteria”

Constraint

You are limited by what is possible (*limits of technology or physics*)

How to relax?

Preference

You are limited by bias expressing preferred tradeoffs

How to relax?

Capability

You are limited by the implementation team (*what they understand, and can deliver; their availability and capabilities*)

How to relax?

Understand Your (System-)Limits

Overspecification is a painful limit, but only one limit.

As architect, you balance many limits:

Requirement

You are limited by mandate (*specification*) for what system must do

How to relax?

- Can: Negotiate the mandate, or change “Acceptance Criteria”

Constraint

You are limited by what is possible (*limits of technology or physics*)

How to relax?

- Can: Explore alternative technology

Preference

You are limited by bias expressing preferred tradeoffs

How to relax?

Capability

You are limited by the implementation team (*what they understand, and can deliver; their availability and capabilities*)

How to relax?

Understand Your (System-)Limits

Overspecification is a painful limit, but only one limit.

As architect, you balance many limits:

Requirement

You are limited by mandate (*specification*) for what system must do

How to relax?

- Can: Negotiate the mandate, or change “Acceptance Criteria”

Constraint

You are limited by what is possible (*limits of technology or physics*)

How to relax?

- Can: Explore alternative technology

Preference

You are limited by bias expressing preferred tradeoffs

How to relax?

- Can: Negotiate different tradeoff ranking

Capability

You are limited by the implementation team (*what they understand, and can deliver; their availability and capabilities*)

How to relax?

Understand Your (System-)Limits

Overspecification is a painful limit, but only one limit.

As architect, you balance many limits:

Requirement

You are limited by mandate (*specification*) for what system must do

How to relax?

- Can: Negotiate the mandate, or change “Acceptance Criteria”

Constraint

You are limited by what is possible (*limits of technology or physics*)

How to relax?

- Can: Explore alternative technology

Preference

You are limited by bias expressing preferred tradeoffs

How to relax?

- Can: Negotiate different tradeoff ranking

Capability

You are limited by the implementation team (*what they understand, and can deliver; their availability and capabilities*)

How to relax?

- Can: Train the team, or use different team

Review: Development Model

Requirements: Is all about system purpose

...So we can address
our Business Need

System Analysis: Is all about technical behavior

...So system purpose
can be achieved

Architecture & Design: Is all about Theory of Operation

...For a chosen
approach

Implementation: Is all about construction and delivery

...To manifest
Theory of Operation

Which of these is optional?

(Which do you want to leave out?)

What is System Development?

The delivery of identified and managed dependencies with respect to limits of requirement, preference, constraint, and capability.

Who manages this?

The Architect.

Tough job?

Yes, look what you are balancing:
Delivery of the right dependencies,
given the reality of your constraints.

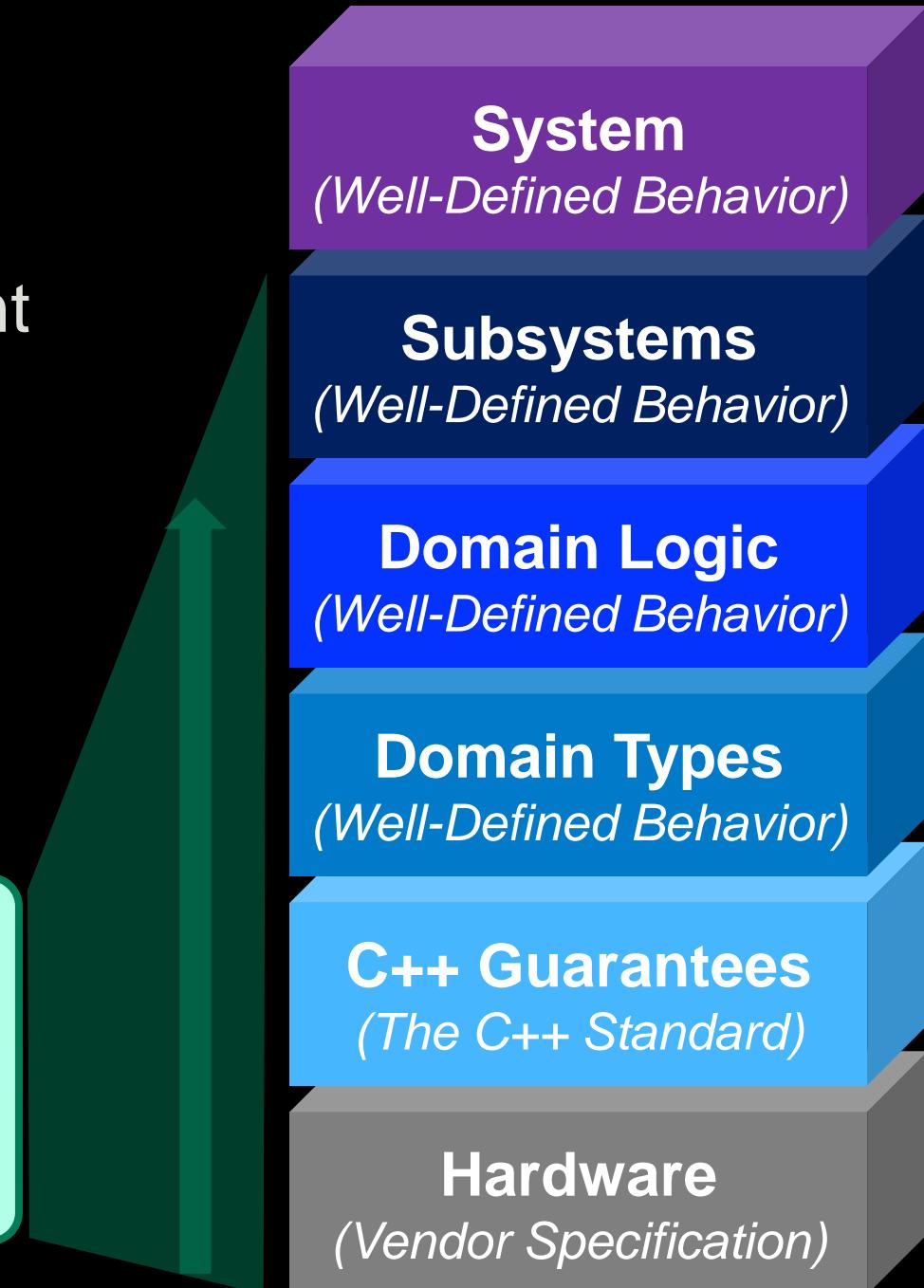
That's
Engineering 101

Using C++ Looks Like...

Because of direct reasoning through the Abstract State Machine, C++ development looks like the following:

“Bottom-Up”: Developer maps from “hardware-up” through C++ Guarantees

1. Define domain types (*from fundamental types*)
2. Implement domain algorithms (*from domain types*)
3. Implement domain subsystems (*from domain algorithms*)



Using C++ Looks Like...

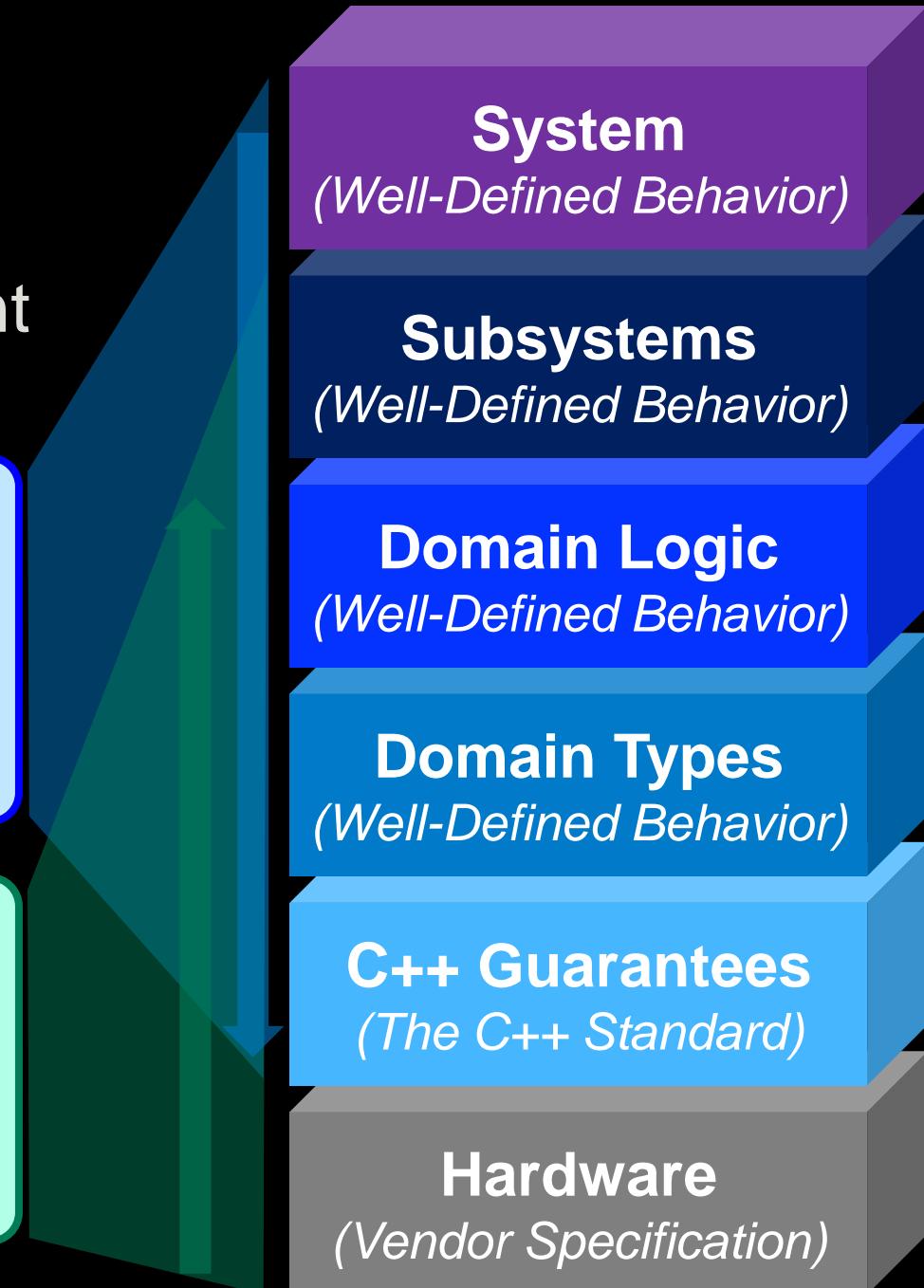
Because of direct reasoning through the Abstract State Machine, C++ development looks like the following:

“Top-Down”: Developer maps from conceptualized system through domain types

1. Define Theory of Operation
2. Implement domain subsystems (*for Theory of Operation*)
3. Implement domain algorithms (*from domain subsystems*)

“Bottom-Up”: Developer maps from “hardware-up” through C++ Guarantees

1. Define domain types (*from fundamental types*)
2. Implement domain algorithms (*from domain types*)
3. Implement domain subsystems (*from domain algorithms*)



Development Axes

Many Degrees Of Freedom

Real-World Development Stages

- What To Expect In Development (*Real-World*):

1 **System Analysis** (*What do you want to build?*)

Real-World Development Stages

CPP-Summit

- What To Expect In Development (*Real-World*):

1

System Analysis (*What do you want to build?*)



No, seriously, what are we being paid to deliver?

Real-World Development Stages

CPP-Summit

- What To Expect In Development (*Real-World*):

1

System Analysis (*What do you want to build?*)



No, seriously, what are we being paid to deliver?

2

Conceptualize

(Define Theory of Operation by contrasting the system you will build against systems you will not build)

Feasibility (*What specific technical questions must be answered before you understand the target system, and you know the system can be built?*)

Prototype (*Demonstrate the system can be viably built*)

1. Can be combined with Feasibility
2. Can be skipped if you have an existing system, and new system uses similar technologies

Real-World Development Stages

CPP-Summit

- What To Expect In Development (*Real-World*):

1

System Analysis (*What do you want to build?*)



No, seriously, what are we being paid to deliver?

2

Conceptualize

(Define Theory of Operation by contrasting the system you will build against systems you will not build)

```
if(prototype_works)
    goto 3;
    goto 2;
```

The only Hard-Barrier!

Feasibility (*What specific technical questions must be answered before you understand the target system, and you know the system can be built?*)

Prototype (*Demonstrate the system can be viably built*)

1. Can be combined with Feasibility
2. Can be skipped if you have an existing system, and new system uses similar technologies

3

Development (*Build It!*)

Design (*Implement Theory of Operation*)

Implement (*Code / Implement Design*)

Validate (*Confirm need is addressed, Verify technical behavior*)

Real-World Development Stages

CPP-Summit

- What To Expect In Development (*Real-World*):

1

System Analysis (*What do you want to build?*)



No, seriously, what are we being paid to deliver?

2

Conceptualize

(Define *Theory of Operation* by contrasting the system you will build against systems you will not build)

```
if(prototype_works)
    goto 3;
    goto 2;
```

The only Hard-Barrier!

Feasibility (*What specific technical questions*

must be answered before you understand the target system, and you know the system can be built?)

Prototype (*Demonstrate the system can be viably built*)

1. Can be combined with Feasibility
2. Can be skipped if you have an existing system, and new system uses similar technologies

3

Development (*Build It!*)

Design (*Implement Theory of Operation*)

Implement (*Code / Implement Design*)

Validate (*Confirm need is addressed, Verify technical behavior*)

4

Deploy (*e.g., Handoff to manufacturing, update user docs, train support staff, make available to customer*)

Feedback from later phases can “inform” earlier phases

“Deploy” (*customer feedback*) can impact System Concept

The Only “Hard Barrier”

The Architect must be the “adult” when considering “sparkly new things”

NEVER “Scale Up” a Feasibility exercise *(pretending it to be “Development”)*

- Leads to:

Massive Cost Increases

(production-level resources on an unproven experiment)

Amplified Risk

(deployment-level blast radius for an unproven experiment)

Quality Drop, Developer Confusion

(mingles lower-standard experimentation with production-level quality expectations)



Parallel Progression

May evolve “in parallel”:

What are you building?

Concept Progression

(evolve across phases):

- Requirements
- Conceptualization
- Development

Surprise!

- Key Customer (\$) redirects effort
- Target use-case changed
- Business focus changed

Perhaps due to
acquisition, buyout, or merger

Perhaps due to
“fast-moving” market

How is the building going?

Development Progression

(evolve across phases):

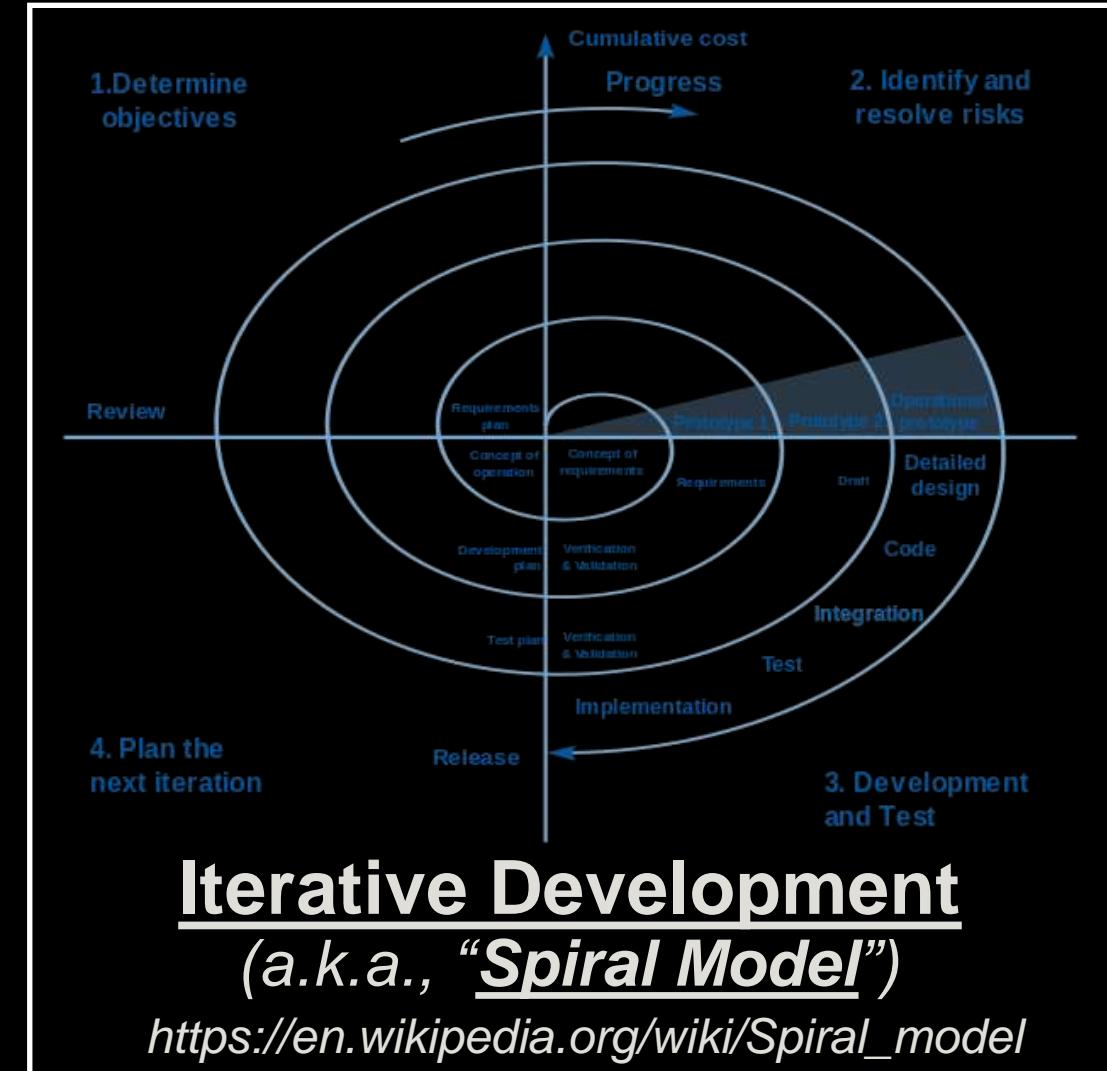
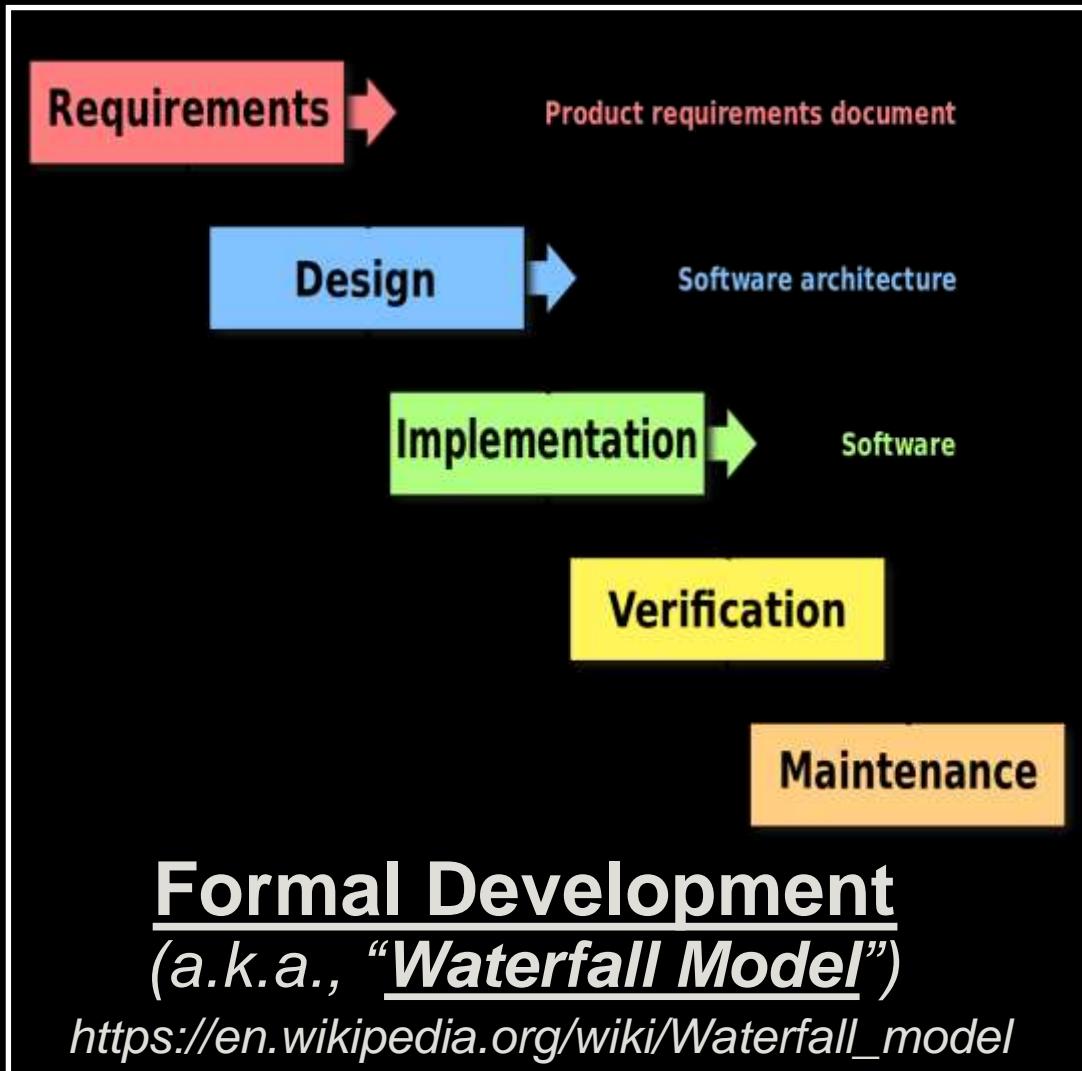
- Feasibility / Prototyping
- Development
- Productization

Surprise!

- Port to new platform / OS
- Port to new compiler / toolchain
- Change / Add 3rd party subsystem

Two Models Exist

- In general, Two (2) Development Models exist:



Development “Space”

We Know
Exactly

Conceptualization
(What Are You Building?)

We Have
No Idea

Development
Space

Haven't
Started

Development
(How Is The Building Going?)

We're
Done!

Development “Space”

We Know
Exactly

Conceptualization
(What Are You Building?)

We Have
No Idea

Haven't
Started

Start Here

Development
(How Is The Building Going?)

We're
Done!

Development
Space

Development “Space”

We Know
Exactly

Conceptualization
(What Are You Building?)

We Have
No Idea

Haven't
Started

Development
(How Is The Building Going?)

We're
Done!

Want to be Here

Start Here

Development
Space

Development “Space”

We Know
Exactly

Conceptualization
(What Are You Building?)

We Have
No Idea

Haven't
Started

Development
(How Is The Building Going?)

We're
Done!

Want to be Here

Start Here

Q: What would “Spiral”
(or “Agile”) do?

Development “Space”

We Know
Exactly

Conceptualization
(What Are You Building?)

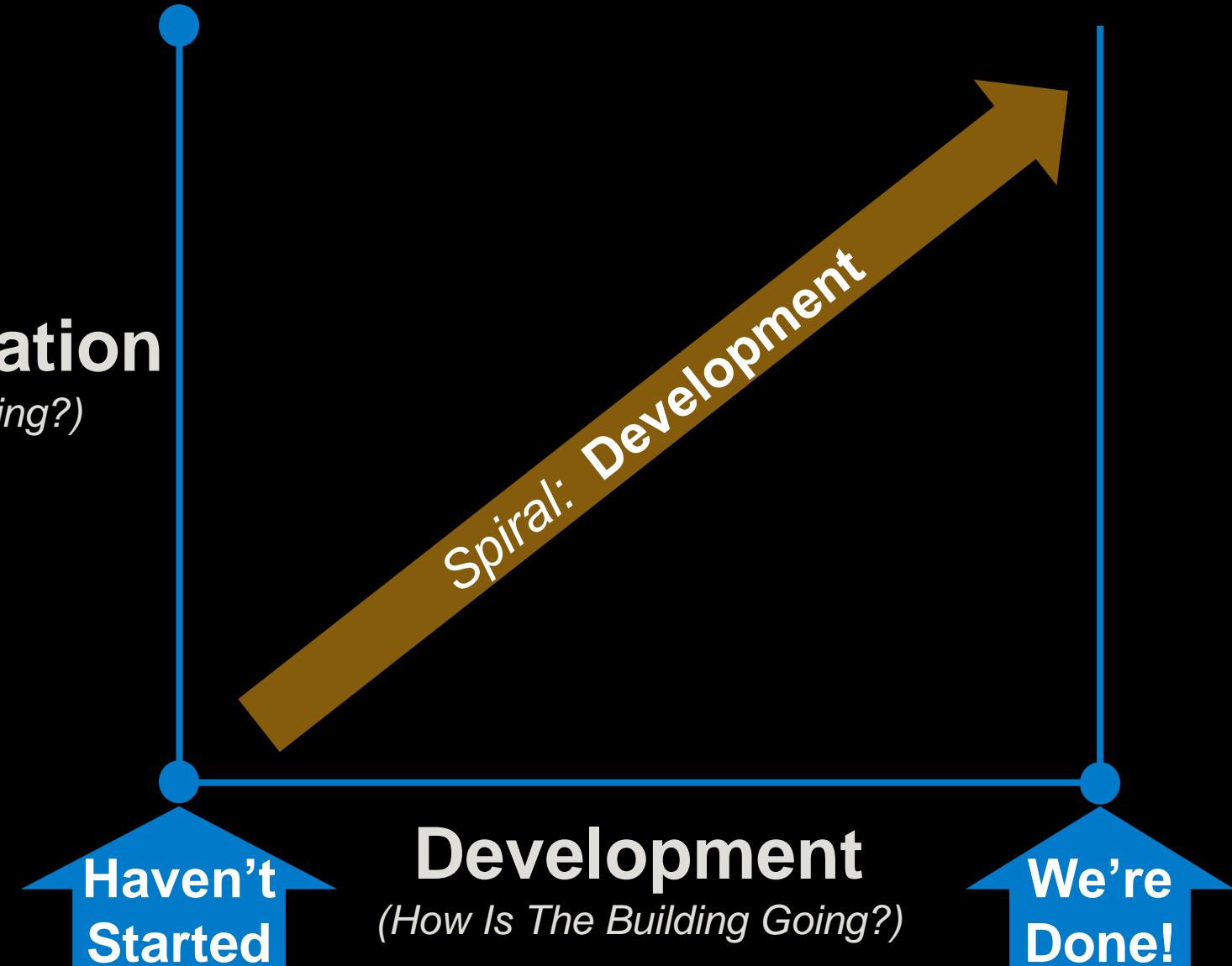
We Have
No Idea

Haven't
Started

Development
(How Is The Building Going?)

We're
Done!

Spiral: Development



Development “Space”

We Know
Exactly

Conceptualization
(What Are You Building?)

We Have
No Idea

Haven't
Started

Development
(How Is The Building Going?)

We're
Done!

Spiral: Development

Q: *What would “Formal”
(or “Waterfall”) do?*

Development “Space”

We Know
Exactly

Conceptualization
(What Are You Building?)

We Have
No Idea

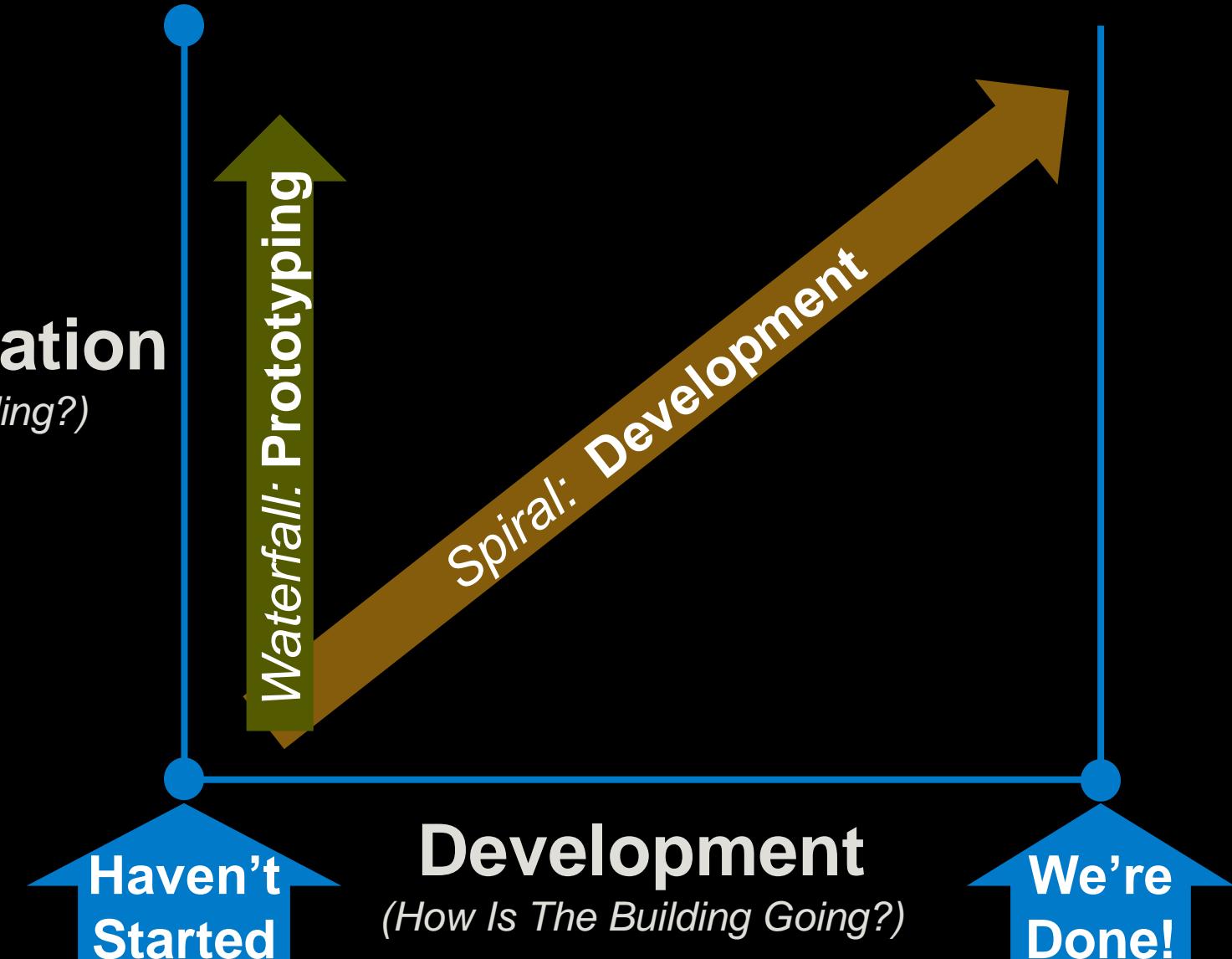
Haven't
Started

Development
(How Is The Building Going?)

We're
Done!

Waterfall: Prototyping

Spiral: Development



Development “Space”

We Know
Exactly

Conceptualization
(What Are You Building?)

We Have
No Idea

Haven't
Started

Development
(How Is The Building Going?)

We're
Done!

Waterfall: Development

Spiral: Development

Waterfall: Prototyping

Architectural “Space”

We Know
Exactly

Conceptualization
(What Are You Building?)

We Have
No Idea

Haven't
Started

Development
(How Is The Building Going?)

We're
Done!

Architectural
Space

**Target
Customer**
*(What Is Needed,
What Do They Want?)*

Could Be (and often is!)
• Multi-Target
• Multi-Dimensional

Architectural “Space”

We Know Exactly

Conceptualization

(What Are You Building?)

We Have No Idea

Haven't Started

Development

(How Is The Building Going?)

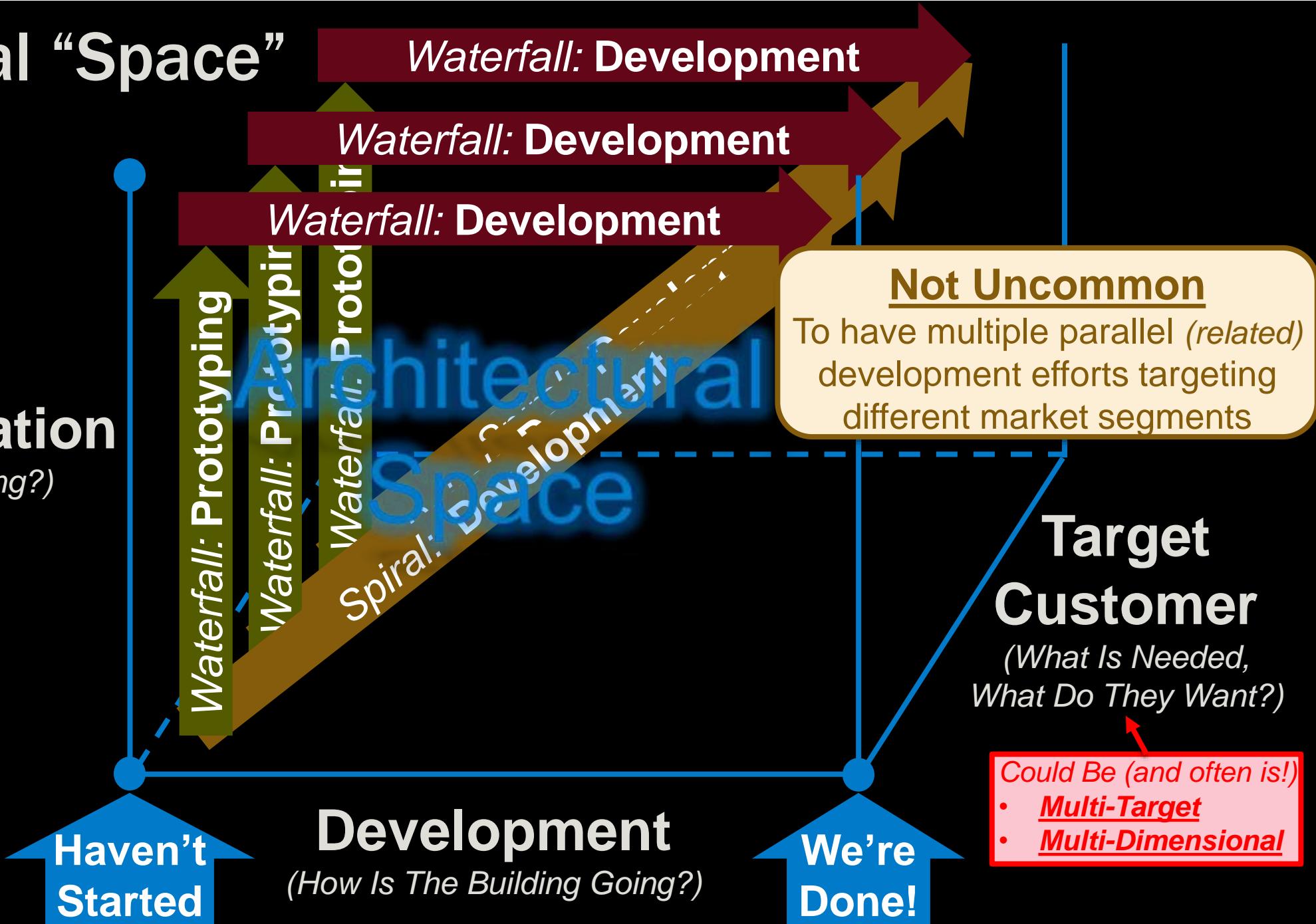
We're
Done!

Target Customer

*(What Is Needed,
What Do They Want?)*

Could Be (and often is!)

- Multi-Target
- Multi-Dimensional



Who Is Found On Each Axis?

We Know
Exactly

Who exists
on this axis?

Conceptualization

(What Are You Building?)

Architectural
Space

We Have
No Idea

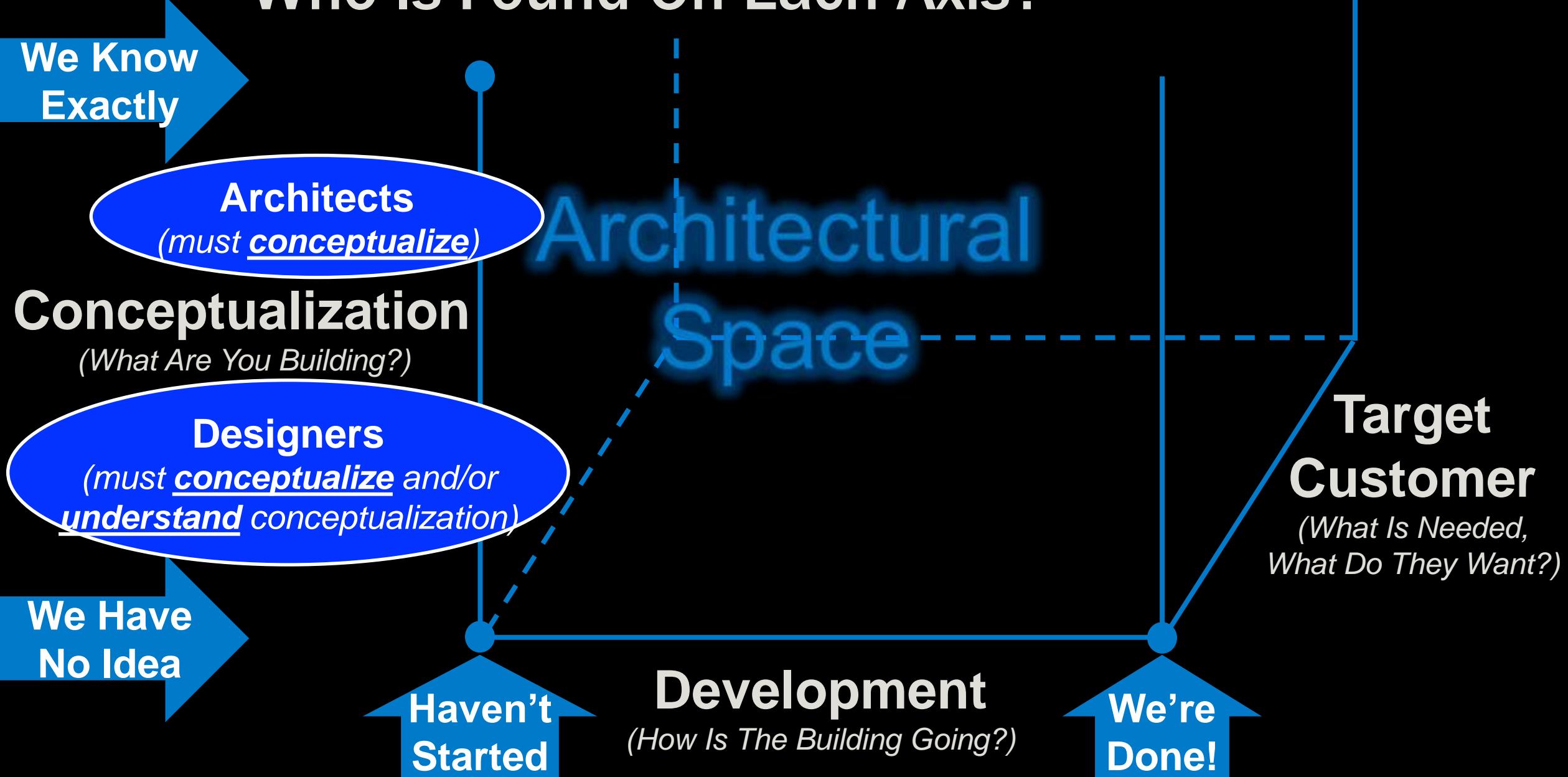
Haven't
Started

Development
(How Is The Building Going?)

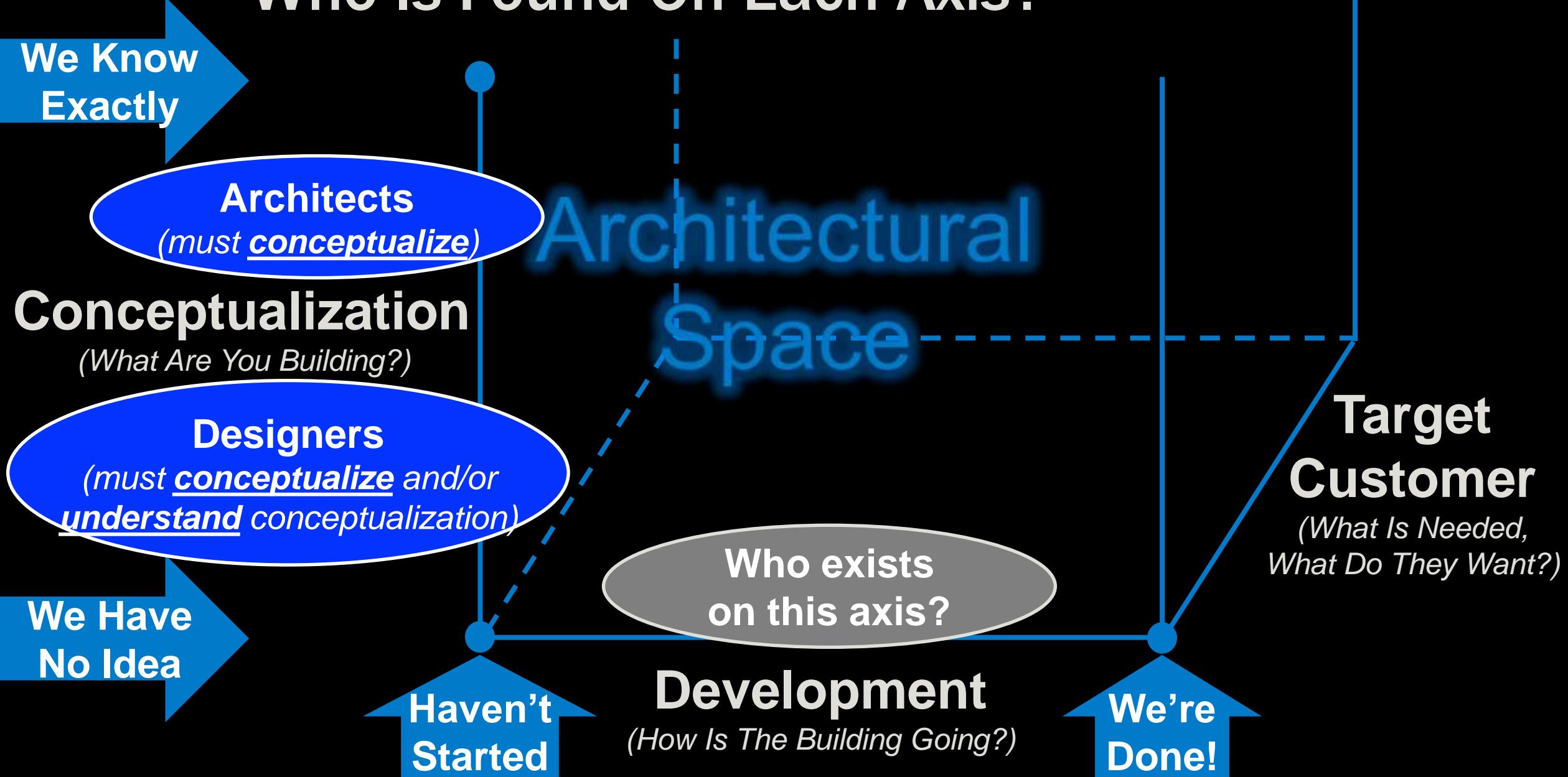
We're
Done!

**Target
Customer**
*(What Is Needed,
What Do They Want?)*

Who Is Found On Each Axis?



Who Is Found On Each Axis?



Who Is Found On Each Axis?

We Know
Exactly

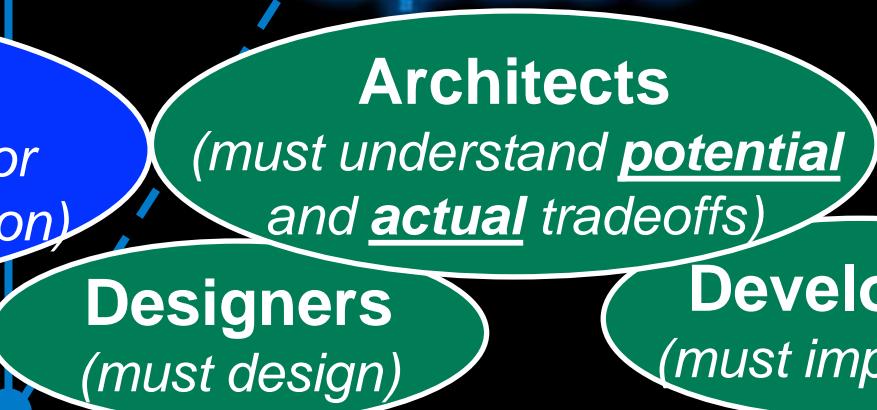


Conceptualization

(What Are You Building?)



We Have
No Idea



Target
Customer
(What Is Needed,
What Do They Want?)

Development
(How Is The Building Going?)

Haven't
Started

We're
Done!

Who Is Found On Each Axis?

We Know
Exactly



Conceptualization
(What Are You Building?)

Designers
(must conceptualize and/or
understand conceptualization)

Architects
(must understand potential
and actual tradeoffs)

Designers
(must design)

Developers
(must implement)

Who exists
on this axis?

**Target
Customer**
*(What Is Needed,
What Do They Want?)*

We Have
No Idea

Haven't
Started

Development
(How Is The Building Going?)

We're
Done!

Who Is Found On Each Axis?

We Know
Exactly

Architects

(must conceptualize)

Conceptualization

(What Are You Building?)

Designers

(must conceptualize and/or
understand conceptualization)

We Have
No Idea

Architects

(must understand potential
and actual tradeoffs)

Designers

(must design)

Developers

(must implement)

Haven't
Started

Development

(How Is The Building Going?)

We're
Done!

Architects

(must understand short-term and
long-term goals, market-position)

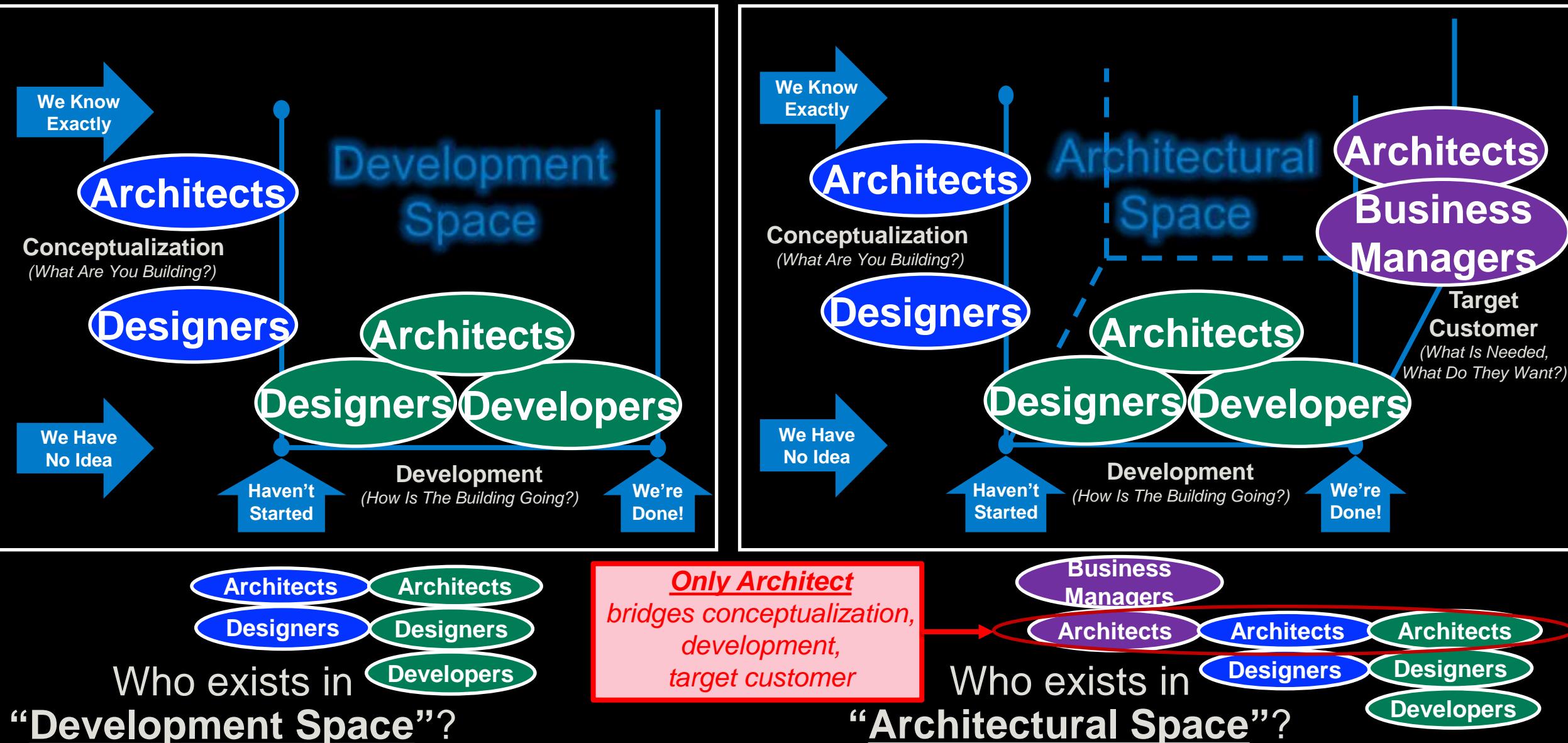
Business Managers

(must define position
within marketplace)

Target
Customer

(What Is Needed,
What Do They Want?)

Development vs. Architectural “Space”



Reuse Across Product Lines

- The more variations in product offering (or target customer), the more motivation for reuse

Product (def): A unit offered for delivery

Product Line (def): A collection of related-but-unique product offerings

Product Family (def): A collection of related-but-unique product lines

Market Gap Analysis (def): Compared placement of your product families within the superset of product families offered in your market segment

Each “related-but-unique” motivates some form of architectural reuse

Physical Dimensions Within Architectural Space

- (Reuse) **What code is...**



Application-Specific

Reusable within the product line

Reusable within the product family

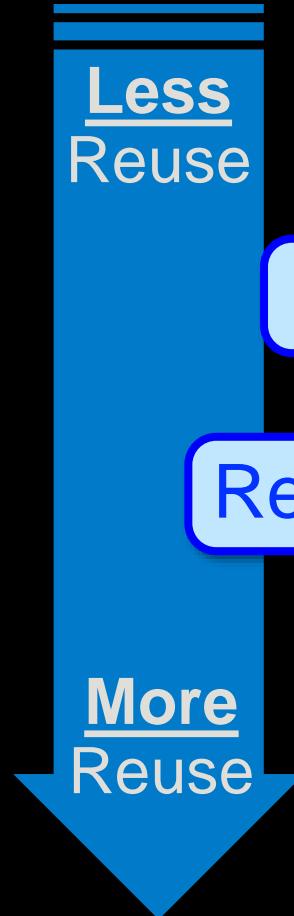
Reusable within the business

Reusable by 3rd Parties

- Specific Considerations:
 - User interface
 - Subsystem Configuration
 - System Configuration
 - Business Logic Invariants:
 - Types
 - Processing Models
 - Control Flows
 - Data Flows
 - Serviceability / Support Interfaces
 - Domain-Specific Data Handling
 - Logging
 - Serialization
 - RPC, Distributed processing models

Physical Dimensions Within Architectural Space

- (Reuse) **What code is...**



- Specific Considerations:
 - User interface
 - Subsystem Configuration
 - System Configuration
 - Business Logic Invariants:
 - Types
 - Processing Models
 - Control Flows
 - Data Flows
 - Serviceability / Support Interfaces
 - Domain-Specific Data Handling
 - Logging
 - Serialization
 - RPC, Distributed processing models

Reuse Has A Cost

- Ties developer progression

- Good: Promotes consistency, “everybody knows how it always works”
- Bad: Forces same solution to different problem, decreases innovation

- Can slow or speed development

- Good: Every Developer benefits from library update
- Bad: Every Developer waits on library update

- Can raise or lower cost

- Good: Lowers cost of new systems, provides economies of scale
- Bad: Cannot “just fix it” in one system, because of behavior impact in other systems↑(higher coupling adds complexity, time, and risk)

Who cares?
Managers

Who cares?
Security Professionals

Who cares?
Innovators

Who cares?
3rd Parties

Who cares?
Big Companies

Who cares?
Support Engineers

Importance varies among
your diverse interested parties
(must rank and balance priorities!)

Mature, Large organizations tend to prioritize for:
Consistency and **Maintainability**

*Thank you!
for listening*

Questions?



Dori Exterman

Incredibuild CTO

Dori 是Incredibuild的首席技术官，他负责指导公司的产品战略，技术架构，并负责产品愿景、实施和技术合作。在加入 Incredibuild 之前，Dori 在多家软件公司担任过各种技术和产品开发角色，专注于架构、性能、先进技术、开发Ops、发布管理和 C++. 他是软件开发技术方面的专家。

主办方：

Boolan
高端IT咨询与教育平台

C++ Summit 2020

Dori Exterman

Incredibuild CTO

4 Software Development Predictions for 2021

C++ Summit 2020

Dori Exterman

Incredibuild CTO

How to Push Enterprises Over the Dev Speed Limit

2020 Agile is no longer a trend

2020 Agile is the standard

HOW MANY STARTUPS USE AGILE?

95% of respondents report their organizations practice agile development methods.

2019

95% 

(From: [State of Agile Report](#))

Advantages of agile for enterprises

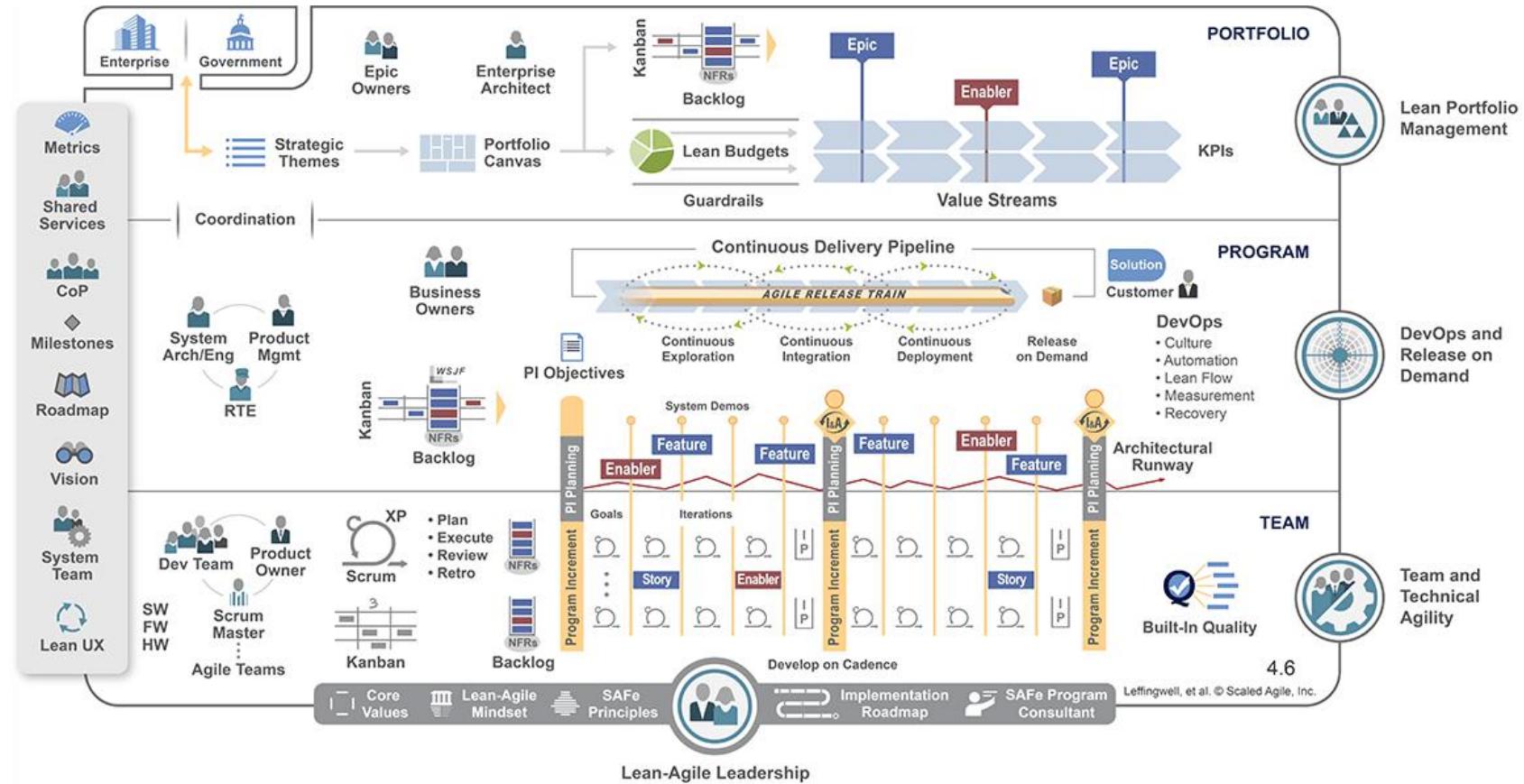
- Faster time to market
- Quicker update release (improved quality)
- More frequent iterations
- Better cross-company collaboration
- Minimize resource waste

Digital is the main reason just over half of the **companies** on the Fortune 500 have disappeared since the year 2000

Pierre Nanterme
CEO of Accenture



Super complex implementation flow:



The main 3 principles of Agile

The Manifesto for Agile Software Development

- Customer satisfaction by **early and continuous delivery** of valuable software.
- Welcome **changing requirements**, even in late development.
- Deliver working software **frequently** (weeks rather than months)

Agile pre-requisites

- CI\CD Automation
- Automated test coverage
- Fast dev cycles
- Frequent iterations

4 speedy predictions for enterprises

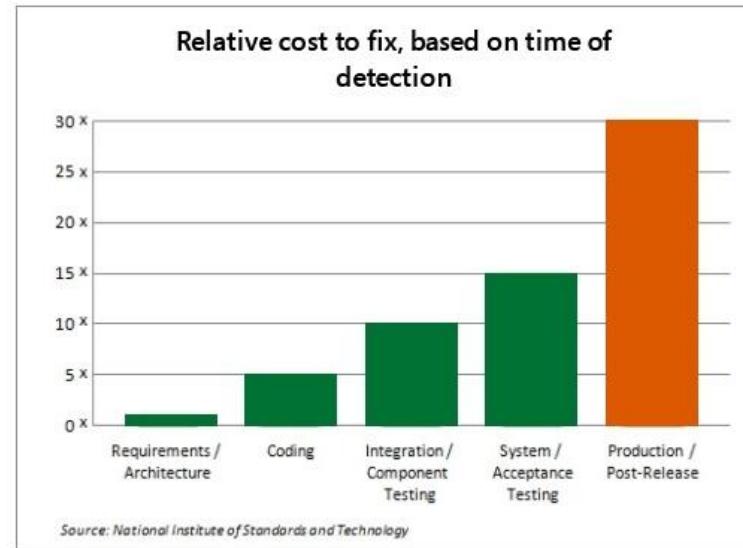
Two mega trends

Shift-left

- find and prevent defects early in the software delivery process
- improve quality by moving tasks to the left as early in the lifecycle as possible

Cloud transformation

- the process of moving your work to the cloud



Chellenges Enterprises are facing Adopting Agile practices

- Code inflation
- Technical debt in testing
- Limites test coverage automation
- Slow and in-frequent dev cycles
 - From nightly builds -> intra-day -> build per commit

Code inflation

“

Over half (51%) of software development stakeholders report they have **more than 100 times the volume of code they had 10 years ago.** And a staggering 18% say they have 500 times more code.

From: [The emergence of big code](#)
(Sourcegraph + Dimensional Research)

Problem #1

Test-coverage technical debt

- Heavy burden on manual QA
- Bugs are found late in the development cycle
- In-frequent releases
- Product quality issues

Prediction #1

**AI-based test generation will make
testing faster and better**

Prediction #1

AI-based test generation:

Harvest Artificial Intelligence automation to catch up with technical debt

- Catch errors, bugs, and regressions early and often
- Streamline CI pipelines, improving lead time and deployment frequency
- Integrate with more confidence and increase product quality



diffblue
AI for Code

The screenshot shows a Java code editor in an IDE. The project structure on the left includes packages like CoreBanking, com.diffblue.corebanking, and com.diffblue.corebanking.compliance.rules. The current file is ComplianceRule.java, which contains the following code:

```
package com.diffblue.corebanking.compliance.rules;

import com.diffblue.corebanking.account.Account;

public class ComplianceRuleLargeBalance extends ComplianceRule {
    /**
     * Checks if the passed account passes or fails this rule.
     *
     * @param account The account to verify compliance.
     */
    @Override
    public void validateAccountCompliance(final Account account) {
    }
}
```

The code editor has syntax highlighting and a code completion dropdown open over the word "CheckCompliance". The bottom right corner of the IDE window features the diffblue logo and the text "AI for Code".

Additional players

- Testim.io
- InteliTTest
- PoniCode

Automated test generation for C++ code is more complex than for intermediate languages (c#, java) or script languages such as java script.

Problem #2

Full test execution can take hours

- Good code coverage can lead to hours of test execution
- Testing each commit can be impossible
- Not testing each commit leads to "who broke the build" long resolution
- Running hours of test frequently is very costly

Prediction #2

Test avoidance tech will make a dent in unnecessary testing

Why avoidance?

Test automation means test inflation:

- Unit tests
 - API
 - Integration tests
 - Regression tests
- 
- End-to-end tests
 - Sanity
 - Fuzzy testing
 - DevSecOps

“

I have a small change.

Why do I need to run all the tests?

- Every developer



Kohsuke Kawaguchi

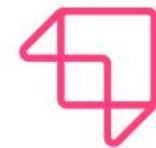


Jenkins



CloudBees.

“ As [software development](#) becomes bigger in scale, this [testing] waste costs millions of dollars and countless hours of developer time.



Launchable

Machine learning in test automation tooling
to identify tests that matter.

 Launchable success story

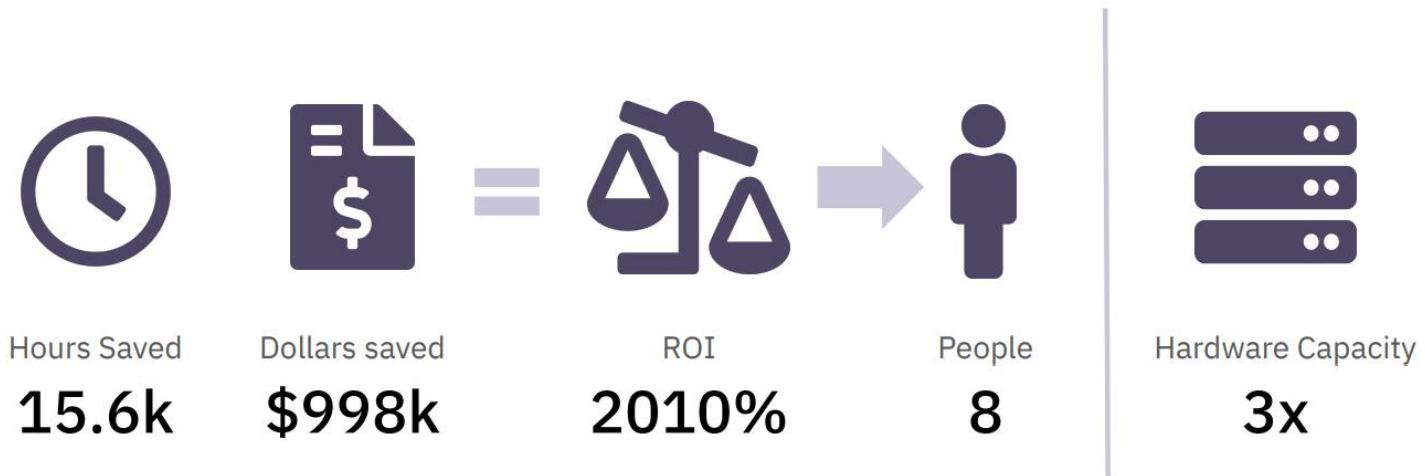
A leading car manufacturer



 Launchable success story

A leading car manufacturer

Impact





**But in the
meantime**

Problem #3

Slow and infrequent build time

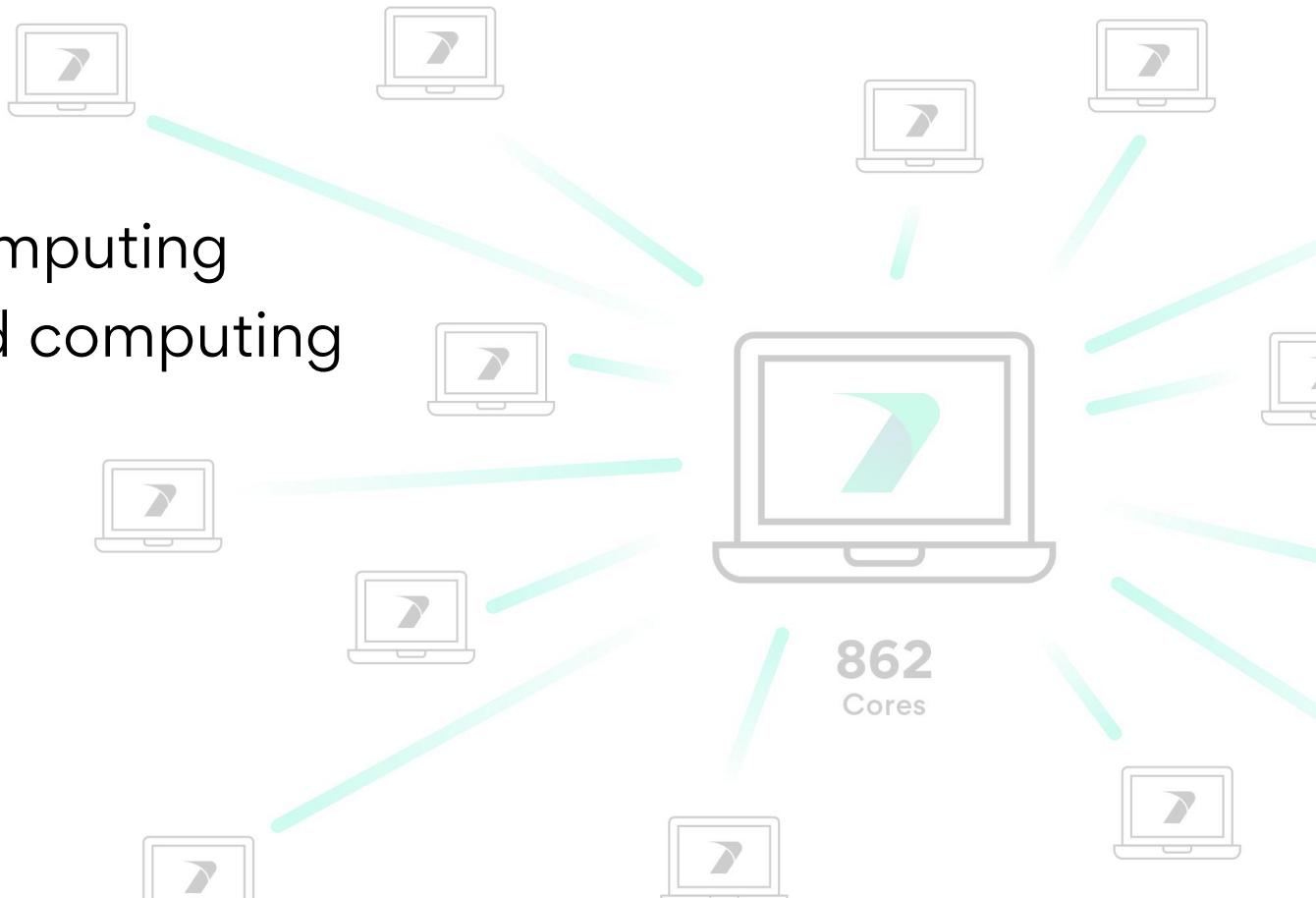
- Large code base
- Time-consuming test execution
- Additional build steps (code analysis, packaging, signing)
- Nightly or periodic builds lead to:
 - Long "who broke the build" resolution process
 - Not having a working version on a daily basis

Prediction #3

High-performance computing

Possible solutions for harvesting more processing power

- HPC
- Cluster computing
- Distributed computing





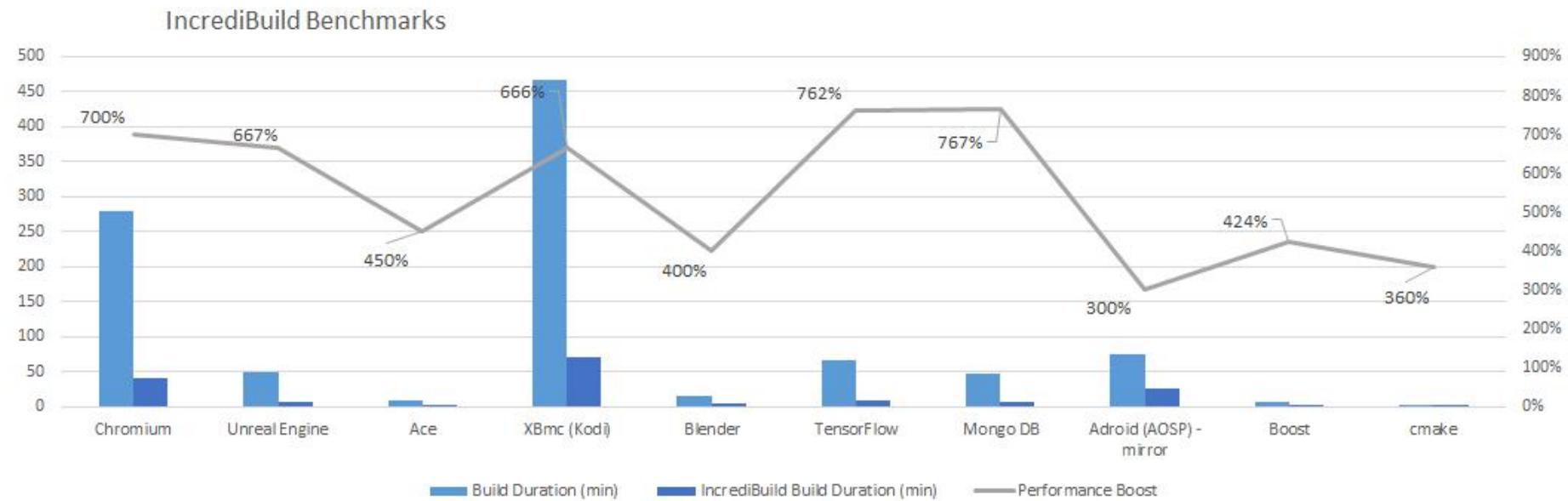
Automated test execution using distributed computing

11 Min

11 Hrs

- Transform each build host into a super computer with hundreds of cores
- Scale-up/down seamlessly and on-demand
- Use simultaneously on-prem and in the cloud
- No IT overhead

C++ GitHub projects compilation accelerated with distributed computing



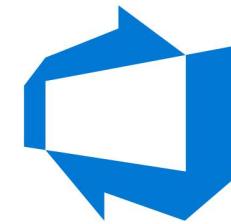
Prediction #4

Managed CI/CD in the cloud will automate release pipelines

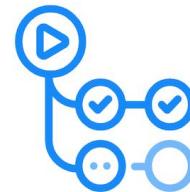
Multiple players in the startup space



AWS CodePipeline



Azure DevOps



GitHub Actions



Benefits for CI/CD in the cloud for enterprises:

- Scale-friendly
- Secure
- Faster onboarding
- IT managed
- Simple repository creation
- Seamless work across geos

Additional cloud-transformation solutions:

- Cloud repositories (GitHub, GitLab, Perforce cloud)
- Cloud builds for developers (GitHub codespaces)
- Virtual Cloud desktops
- SaaS testing – AppliTools
- Work from home bandwidth optimization – Teradici
- And more...



**Before you push your enterprise
over the speed limit...**

Assess your resources and priorities

Enterprise ≠ Slow



VOLVO

3M

 **BOEING**

C++ Summit 2020

Dori Exterman

Incredibuild CTO

Thank you!
dori.exterman@incredibuild.com

CI/CD in the cloud

Honed by startups – adopted by enterprises



“ What's different with cloud is we can deliver business value **at a speed never achieved before.**



Ivan Čukić

《C++函数式编程》作者，KDAB高级工程师

Ivan Čukić 是《C++函数式编程》一书的作者，KDAB高级工程师，同时是最大的C++自由软件项目KDAB的核心开发者，他也是自由软件的热爱者。他还在贝尔格莱德的数学系教授C++与函数式编程。他也是在贝尔格莱德大学获得他的博士学位。他经常受世界各地的技术会议邀请发表演讲。

主办方：

Boolan
高端IT咨询与教育平台



KDAB

Don't sacrifice the API to speed

C++ Summit 2020, China

dr Ivan Čukić

About me

- KDAB senior software engineer
Software Experts in Qt, C++ and 3D / OpenGL
- Author of the "Functional Programming in C++" book
available in English, Chinese, Korean, Russian, Polish
- Trainer / consultant
- KDE developer
- University lecturer



Disclaimer



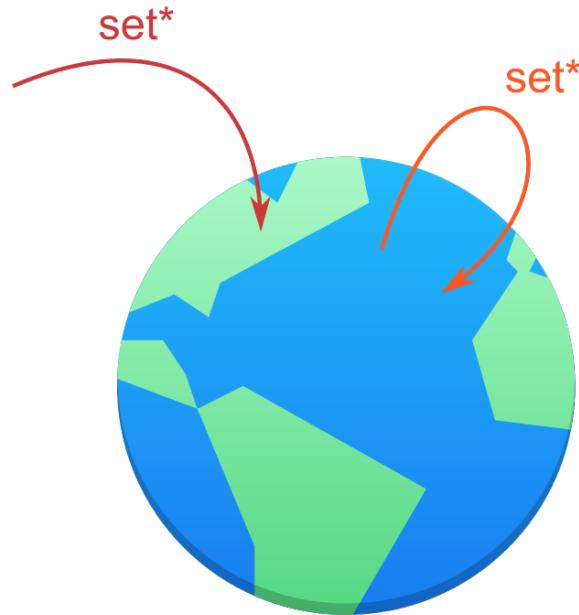
Make your code readable. Pretend the next person who looks at your code is a psychopath and they know where you live.

Philip Wadler

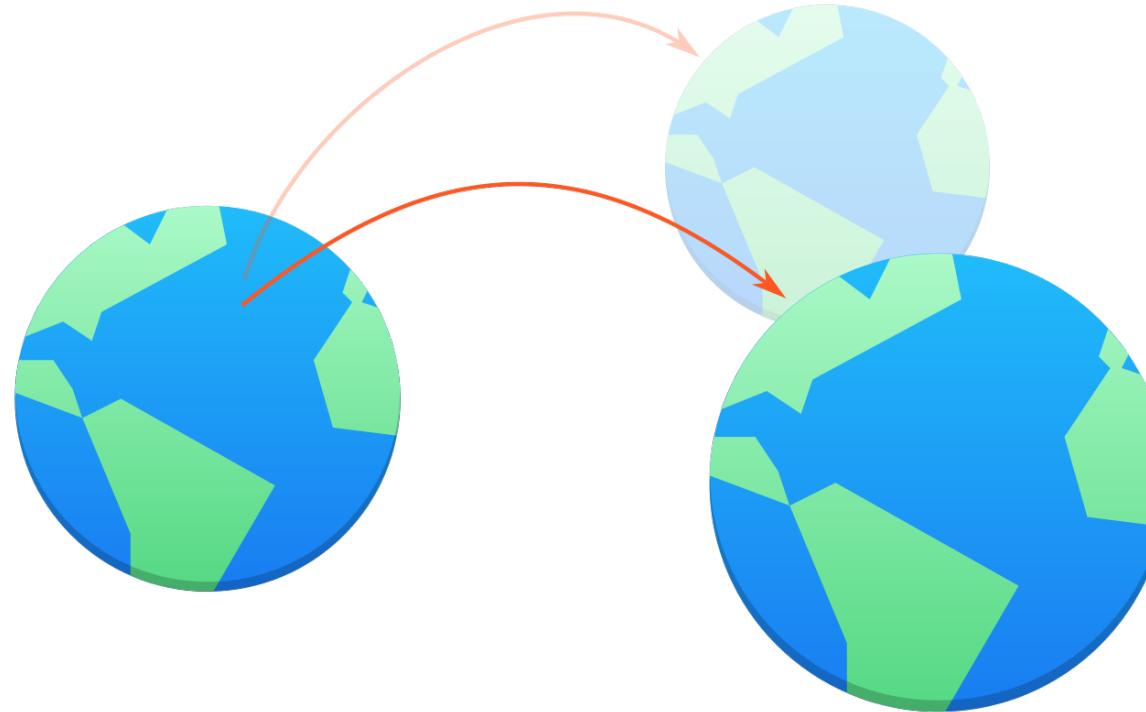
The background of the slide features a dark blue color with a subtle, abstract geometric pattern of light blue triangles of varying sizes and orientations, creating a sense of depth and motion.

FAR AWAY WORLDS

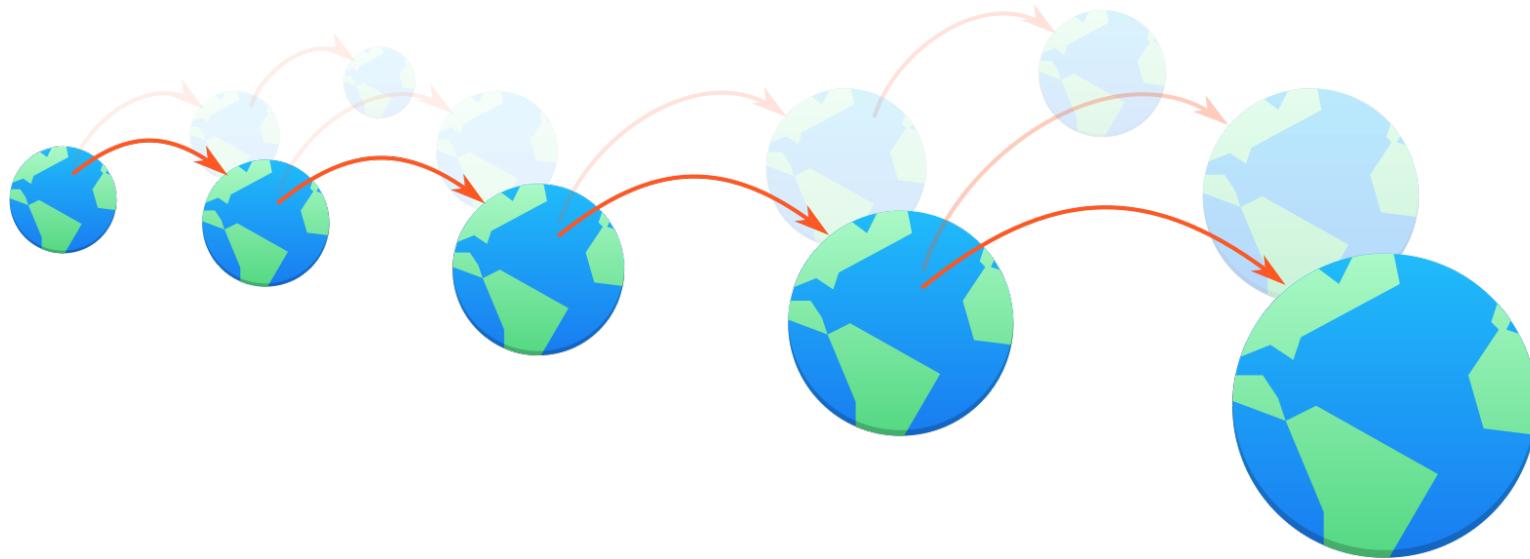
Far away worlds



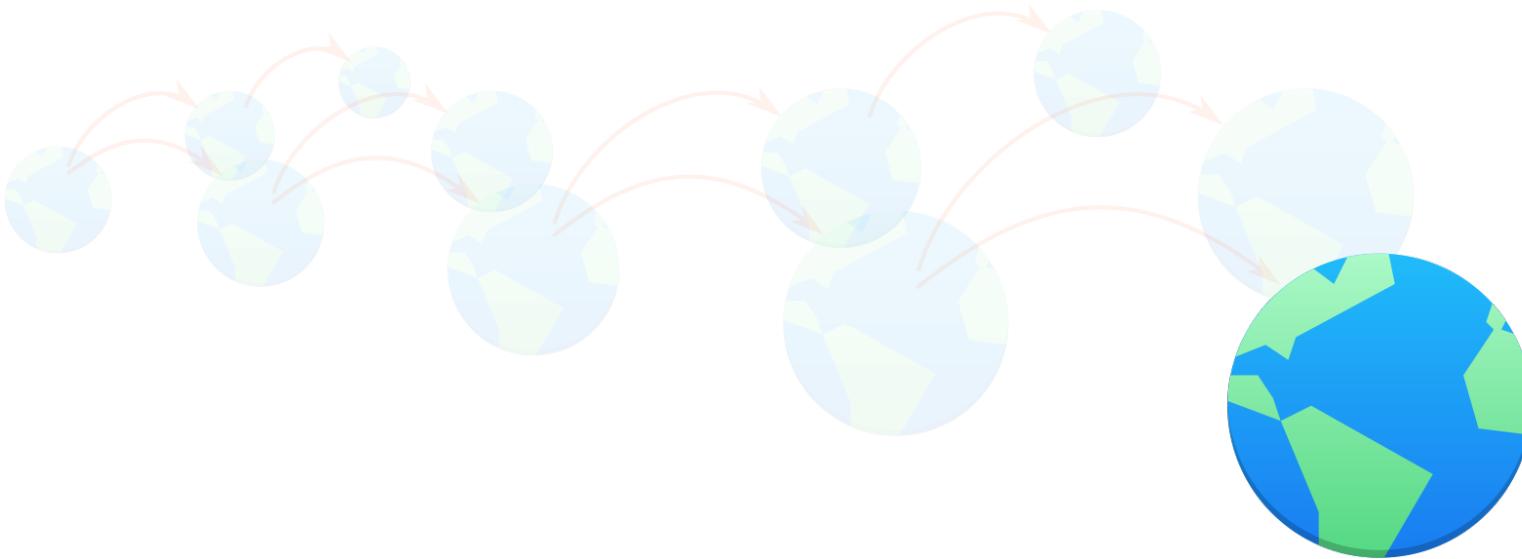
Far away worlds



Far away worlds



Far away worlds



Far away worlds

Values belonging to a linear type must be **used exactly once**: like the world, they can not be duplicated or destroyed. Such values require no reference counting or garbage collection...

Linear types can change the world!
Philip Wadler



ATTACK OF THE CLONES

Far away worlds
ooo

Attack of the Clones
o●oooooooooooo

Generic
oooooooooooooooooooo

Performance
oooooooooooooooooooo

Linear in C++
oooooooooooooooooooo

The End
o

RAII

}





Far away worlds
ooo

Attack of the Clones
ooo●oooooooooooo

Generic
oooooooooooooooooooo

Performance
oooooooooooooooooooo

Linear in C++
oooooooooooooooooooo

The End
o

Clones



Far away worlds
ooo

Attack of the Clones
ooo●oooooooooooo

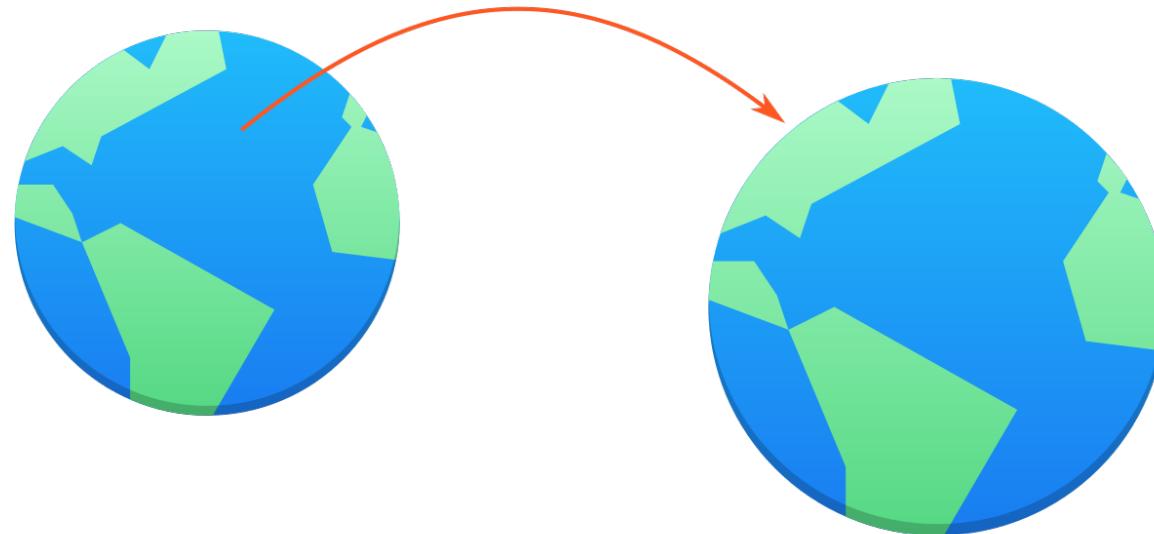
Generic
oooooooooooooooooooo

Performance
oooooooooooooooooooo

Linear in C++
oooooooooooooooooooo

The End
o

Clones



Far away worlds
ooo

Attack of the Clones
ooo●oooooooooooo

Generic
oooooooooooooooooooo

Performance
oooooooooooooooooooo

Linear in C++
oooooooooooooooooooo

The End
o

Clones



Far away worlds
ooo

Attack of the Clones
oooooooo●oooooooooooo

Generic
oooooooooooooooooooo

Performance
oooooooooooooooooooo

Linear in C++
oooooooooooooooooooo

The End
o

Clones

&&



Far away worlds
ooo

Attack of the Clones
oooooooo●oooooooo

Generic
oooooooooooooooooooo

Performance
oooooooooooooooooooo

Linear in C++
oooooooooooooooooooo

The End
o

Clones



Far away worlds
ooo

Attack of the Clones
oooooooo●oooooooo

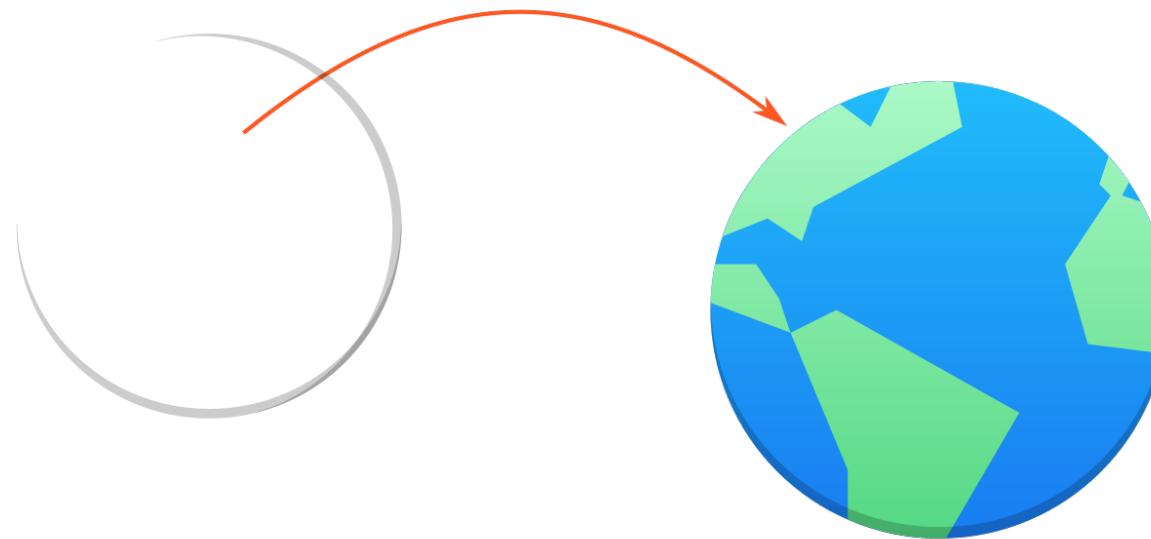
Generic
oooooooooooooooooooo

Performance
oooooooooooooooooooo

Linear in C++
oooooooooooooooooooo

The End
o

Clones



Clones

Move semantics:

- Resource ownership transfer
- Optimization
- API documentation / usage restriction



Clones

Move semantics:

- Resource ownership transfer
- Optimization
- API documentation / usage restriction



Clones

```
void foo(type&& v)
{
    ...
}
```



Clones

```
class type {  
    void foo( ) && | *this is a temporary  
    {  
        ...  
    }  
}
```



Far away worlds
ooo

Attack of the Clones
oooooooo●ooo

Generic
oooooooooooo

Performance
oooooooooooo

Linear in C++
oooooooooooo

The End
o

Clones

```
type&& foo( )
{
    ...
}
```



Clones

```
type&& foo(type&& v)
{
    ...
}
```



Far away worlds
ooo

Attack of the Clones
oooooooooooo●o

Generic
oooooooooooooooooooo

Performance
oooooooooooooooooooo

Linear in C++
oooooooooooooooooooo

The End
o

Clones

```
std::getline(std::cin, s);
```



Clones

```
std::string&& getline(std::istream& in, std::string&& buf);  
  
s = getline(std::cin, std::move(s));
```



GENERIC

Concepts and constraints

How to enforce moves with generic programming?

```
template <typename T>
void foo(T&& val)
{
    ...
}
```



Concepts and constraints

```
template <typename T>
constexpr bool is_int_v = std::is_same_v<T, int>;
```



Concepts and constraints

```
template <typename T>
concept IsInt = std::is_same_v<T, int>;
| not a proper concept,
| demonstration purposes only!
```

Concepts and constraints

```
template <typename T>
    requires (IsInt<T>)
void foo(T&& v)
{
    ...
}
```



Concepts and constraints

```
template <typename T>
    requires (is_int_v<T>)
void foo(T&& v)
{
    ...
}
```



Clones

```
template <typename T>
    requires ( ??? )
void foo(T&& v)
{
    ...
}
```



Clones

```
typedef T& lref;  
typedef T&& rref;
```

```
T value;
```

```
lref& r1 = value; // type of r1 is T&  
lref&& r2 = value; // type of r2 is T&  
rref& r3 = value; // type of r3 is T&  
rref&& r4 = T(); // type of r4 is T&&
```

Clones

```
template <typename T>
    requires (!std::is_lvalue_reference_v<T>)
void foo(T&& v)
{
    ...
}
```



Attack of the clones

```
istream_sequence<std::string> in{std::cin};  
  
std::string result;  
for (const auto& token: in) {  
    result.append(token);  
}
```



Attack of the clones

```
istream<std::string> in{std::cin};  
  
std::string result;  
for (const auto& token: in) {  
    result.append(token);  
}
```

Remember what Sean said?



Attack of the clones

```
istream<std::string> in{std::cin};  
  
const auto result =  
    accumulate(in, string{});
```

Remember what Sean said?

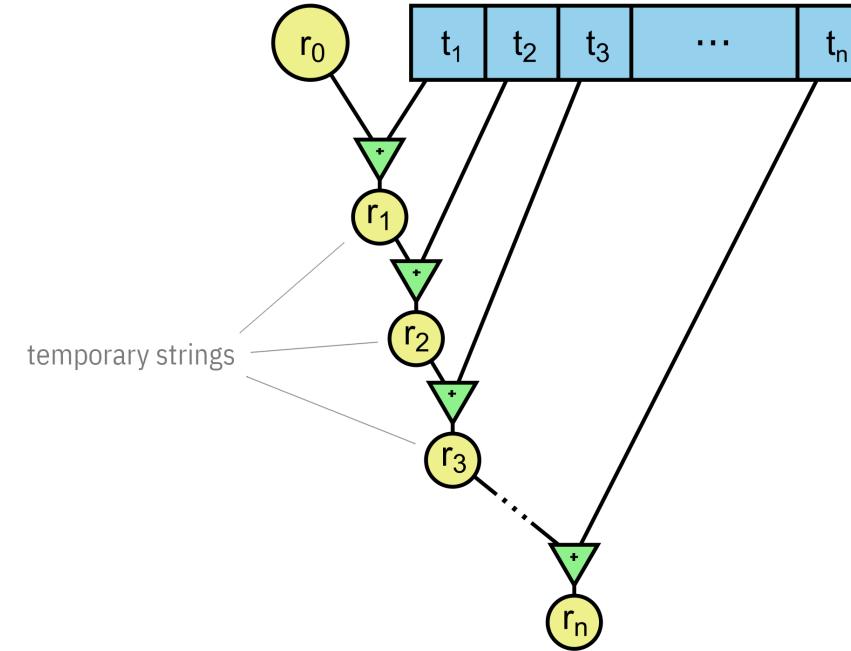


Attack of the clones

```
template <typename InputIt, typename T>
T accumulate(InputIt first, InputIt last, T init)
{
    while (first != last) {
        init = init + *first;
        ++first;
    }
    return init;
}
```



Attack of the clones



Attack of the clones

```
template <typename InputIt, typename T>
T accumulate(InputIt first, InputIt last, T init)
{
    while (first != last) {
        init = std::move(init) + *first;
        ++first;
    }
    return init;
}
```



Far away worlds
ooo

Attack of the Clones
oooooooooooooo

Generic
oooooooooooooo●oooo

Performance
oooooooooooooo

Linear in C++
oooooooooooooo

The End
o

Attack of the clones

Copying is the silent (performance) killer



Far away worlds
ooo

Attack of the Clones
oooooooooooooo

Generic
oooooooooooooooo●oooo

Performance
oooooooooooooooo

Linear in C++
oooooooooooooooo

The End
o

Move-only types

Can we enforce linearity?



Move-only types

- For unit testing generic code
- For message passing, ranges, reactive streams
- For compile-time type tagging



Testing generic code

```
std::accumulate(  
    std::cbegin(items), std::cend(items),  
    std::make_unique<std::string>("Hello, Italian C++!")  
    ...  
) ;
```



Ranges and reactive streams

```
auto pipeline =
    voy::system_cmd("ping"s, "localhost"s)
    | voy::transform([] (lstring value) {
        std::transform(value.begin(), value.end(), value.begin(), toupper);
        return value;
    })
    | append_pid

    | voy::transform([] (lstring value) {
        const auto pos = value.find_last_of('=');
        return std::make_pair(std::move(value), pos);
    })
    | voy::transform([] (std::pair<lstring, size_t>&& pair) {
        auto [ value, pos ] = pair;
        return pos == std::string::npos
            ? std::move(value)
            : std::string(value.cbegin() + pos + 1, value.cend());
    })
    | append_pid

    | voy::filter([] (lstring value) {
        return value < "0.145"s;
    })
    | append_pid

    | voy::sink{cout};
```



CTTT

```
template <typename... NodeMeta>
class node {

    template <typename Meta>
    auto with_meta( ) && | need to move from *this
    {
        return node<Meta, NodeMeta...>(std::move(*this));
    }

};
```



PERFORMANCE

```
1 #include <string>
2 #include <vector>
3
4 std::string f()
5 {
6     std::string s{"Hello"};
7
8     return std::move(s).append(", world!");
9 }
```

A □ 11010 ☑ .LX0: ☐ lib.f: ☑ .text ☑ // ☐ \s+ ☑ Intel ☑ Demangle

Libraries ▾ + Add new... ▾ ⚙ Add tool... ▾

```
1 f[abi:cxx11]():
2     mov    DWORD PTR [rsp-24], 1819043144
3     lea    rdx, [rdi+16]
4     mov    rax, rdi
5     movabs rsi, 2406167339674837036
6     mov    QWORD PTR [rsp-19], rsi
7     mov    BYTE PTR [rsp-20], 111
8     mov    rcx, QWORD PTR [rsp-24]
9     mov    QWORD PTR [rdi], rdx
10    mov    QWORD PTR [rdi+16], rcx
11    mov    ecx, DWORD PTR [rsp-16]
12    mov    QWORD PTR [rdi+8], 13
13    mov    DWORD PTR [rdi+24], ecx
14    movzx  ecx, BYTE PTR [rsp-12]
15    mov    BYTE PTR [rdi+29], 0
16    mov    BYTE PTR [rdi+28], cl
17    ret
```

C Output (0/0) x86-64 gcc 8.3 - 1122ms (455011B)

```

8     linear_wrapper(1&& value)
9     : m_value{std::move(value)}
10    {}
11
12    template <typename... Args>
13    linear_wrapper(std::in_place_t, Args&&... args)
14    : m_value(std::forward<Args>(args)...)
15    {}
16
17    linear_wrapper(linear_wrapper&&) = default;
18    linear_wrapper& operator=(linear_wrapper&&) = default;
19
20    linear_wrapper(const linear_wrapper&) = delete;
21    linear_wrapper& operator=(const linear_wrapper&) = delete;
22
23    inline
24    [[nodiscard]]
25    T&& get() &&
26    {
27        return std::move(m_value);
28    }
29
30
31 private:
32     T m_value;
33 };
34
35 std::string f()
36 {
37     linear_wrapper<std::string> s{std::in_place, "Hello"};
38
39     return std::move(s).get().append(", world!");
40 }
41
42

```

Assembly output for function f:

```

1 f[abi:cxx11]():
2     mov    DWORD PTR [rsp-24], 1819043144
3     lea    rdx, [rdi+16]
4     mov    rax, rdi
5     movabs rsi, 2406167339674837036
6     mov    QWORD PTR [rsp-19], rsi
7     mov    BYTE PTR [rsp-20], 111
8     mov    rcx, QWORD PTR [rsp-24]
9     mov    QWORD PTR [rdi], rdx
10    mov    QWORD PTR [rdi+16], rcx
11    mov    ecx, DWORD PTR [rsp-16]
12    mov    QWORD PTR [rdi+8], 13
13    mov    DWORD PTR [rdi+24], ecx
14    movzx  ecx, BYTE PTR [rsp-12]
15    mov    BYTE PTR [rdi+29], 0
16    mov    BYTE PTR [rdi+28], cl
17    ret

```

Output window:

```

C Output (0/4) x86-64 gcc 8.3 - 1342ms (463603B)

```

Far away worlds
ooo

Attack of the Clones
oooooooooooooo

Generic
oooooooooooooooooooo

Performance
ooo●oooooooooooo

Linear in C++
oooooooooooooooooooo

The End
o

Testing strings

Better than RVO?

/tongue-in-cheek/

Value Proposition: *Allocator-Aware (AA) Software*

John Lakos

Saturday, April 13, 2019

This version is for ACCU'19.



1

```
1 #include <string>
2
3 inline|
4 std::string bin(std::string val) {
5     val.append("Hello C++ !");
6     return val;
7 }
8
9
10 std::string goo(std::string s) {
11     return bin(bin(bin(bin(bin(std::move(s))))));
12 }
```

|

A □ 11010 □ .LX0: □ lib.f: □ .text □ // □ \s+ □ Intel □ Demangle

Libraries □ + Add new... □ Add tool... □

```
1 .LC0:
2     .string "Hello C++ !"
3 bin(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>>)
4     push    r12
5     mov     r12, rdi
6     push    rbp
7     mov     rbp, rsi
8     mov     esi, OFFSET FLAT:.LC0
9     push    rax
10    mov    rdi, rbp
11    call   std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>> __ZNKstd::__cxx11::basic_stringIcharTSt11char_traitsIcharEEc
12    mov    rsi, rbp
13    mov    rdi, r12
14    call   std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>> __ZNKstd::__cxx11::basic_stringIcharTSt11char_traitsIcharEEc
15    mov    rax, r12
16    pop    rdx
17    pop    rbp
18    pop    r12
19    ret
20 goo(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>>)
21     push    r12
22     mov     r12, rdi
23     push    rbp
24     sub    rsp, 168
25     mov     rdi, rsp
26     call   std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>> __ZNKstd::__cxx11::basic_stringIcharTSt11char_traitsIcharEEc
27     mov     rsi, rsp
28     lea     rdi, [rsp+32]
29     call   bin(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>>)
30     lea     rsi, [rsp+32]
```

C

Output (0/0) x86-64 gcc (trunk) - 1448ms (212276B)

```
1 #include <string>
2
3 inline|
4 std::string bin(std::string val) {
5     val.append("Hello C++!");
6     return val;
7 }
8
9
10 std::string goo(std::string s) {
11     return bin(bin(bin(bin(std::move(s))))));
12 }
```

A □ 11010 .LX0: lib.f: .text // \s+ Intel Demangle

Libraries ▾ + Add new... ▾ ⚙ Add tool... ▾

52	53	mov rax, r12
54	55	pop rbp
55	56	pop r12
56	57	ret
57	58	mov rbp, rax
58	59	lea rdi, [rsp+128]
59	60	call std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::basic_string(rdi)
60	61	jmp .L5
61	62	mov rbp, rax
62	.L5:	
63	64	lea rdi, [rsp+96]
64	65	call std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::basic_string(rdi)
65	66	jmp .L6
66	67	mov rbp, rax
67	.L6:	
68	69	lea rdi, [rsp+64]
69	70	call std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::basic_string(rdi)
70	71	jmp .L7
71	72	mov rbp, rax
72	.L7:	
73	74	lea rdi, [rsp+32]
74	75	call std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::basic_string(rdi)
75	76	jmp .L8
76	77	mov rbp, rax
77	.L8:	
78	79	mov rdi, rsp
79	80	call std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::basic_string(rdi)
80	81	mov rdi, rbp
81		call _Unwind_Resume

C Output (0/0) x86-64 gcc (trunk) - 1448ms (212276B)

Returning values

- (N)RVO – result is constructed in the caller
- Moved to the caller (CWG 1579)
- Copied into the caller



CWG 1579

Currently the conditions for moving from an object returned from a function are tied closely to the criteria for copy elision, which requires that the type of the object being returned be the same as the return type of the function. Another possibility that should be considered is to allow something like

```
optional<T> foo() {  
    T t;  
    ...  
    return t;  
}
```

and allow `optional<T>::optional(T&&)` to be used for the initialization of the return type. **Currently this can be achieved explicitly by use of std::move, but it would be nice not to have to remember to do so.**

Returning values

```
U fun( )
{
    T value;
    ...
    return value; // move constructed
}
```

```
1 #include <string>
2
3 inline
4 void bin(std::string& val) {
5     val.append("Hello C++!");
6 }
7
8
9 void goo(std::string& s) {
10    bin(s);
11    bin(s);
12    bin(s);
13    bin(s);
14    bin(s);
15 }
```

A □ 11010 □ .LX0: □ lib.f: □ .text □ // □ \s+ □ Intel □ Demangle

Libraries □ + Add new... □ Add tool... □

```
1 .LC0:
2     .string "Hello C++!"
3 goo(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>&)
4     push  rbp
5     mov   esi, OFFSET FLAT:.LC0
6     mov   rbp, rdi
7     call  std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>&
8     mov   rdi, rbp
9     mov   esi, OFFSET FLAT:.LC0
10    call  std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>&
11    mov   rdi, rbp
12    mov   esi, OFFSET FLAT:.LC0
13    call  std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>&
14    mov   rdi, rbp
15    mov   esi, OFFSET FLAT:.LC0
16    call  std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>&
17    mov   rdi, rbp
18    mov   esi, OFFSET FLAT:.LC0
19    pop   rbp
20    jmp   std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>&
```

C □ Output (0/0) x86-64 gcc (trunk) - 1377ms (190951B)

```
1 #include <string>
2
3 inline
4 std::string&& bin(std::string&& val) {
5     val.append("Hello C++!");
6     return std::move(val);
7 }
8
9
10 std::string&& goo(std::string&& s) {
11     return bin(bin(bin(bin(bin(std::move(s))))));
12 }
```

A □ 11010 ☑ .LX0: ☑ lib.f: ☑ .text ☑ // ☑ ls+ ☑ Intel ☑ Demangle

Libraries ▾ + Add new... ▾ ⚙ Add tool... ▾

```
1 .LC0:
2     .string "Hello C++!"
3 goo(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>&)
4     push    r12
5     mov     esi, OFFSET FLAT:.LC0
6     mov     r12, rdi
7     call   std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>&
8     mov     rdi, r12
9     mov     esi, OFFSET FLAT:.LC0
10    call  std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>&
11    mov     rdi, r12
12    mov     esi, OFFSET FLAT:.LC0
13    call  std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>&
14    mov     rdi, r12
15    mov     esi, OFFSET FLAT:.LC0
16    call  std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>&
17    mov     rdi, r12
18    mov     esi, OFFSET FLAT:.LC0
19    call  std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>&
20    mov     rax, r12
21    pop    r12
22    ret
```

C Output (0/0) x86-64 gcc (trunk) - 1203ms (191834B)

```
1 #include <string>
2
3 inline|
4 std::string bin(std::string val) {
5     val.append("Hello C++!");
6     return val;
7 }
8
9
10 std::string goo(std::string s) {
11     return bin(bin(bin(bin(std::move(s))))));
12 }
```

A □ 11010 .LX0: lib.f: .text // \s+ Intel Demangle

Libraries ▾ + Add new... ▾ ⚙ Add tool... ▾

52	53	mov rax, r12
54	55	pop rbp
55	56	pop r12
56	57	ret
57	58	mov rbp, rax
58	59	lea rdi, [rsp+128]
59	60	call std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::basic_string(rdi)
60	61	jmp .L5
61	62	mov rbp, rax
62	.L5:	
63	64	lea rdi, [rsp+96]
64	65	call std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::basic_string(rdi)
65	66	jmp .L6
66	67	mov rbp, rax
67	.L6:	
68	69	lea rdi, [rsp+64]
69	70	call std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::basic_string(rdi)
70	71	jmp .L7
71	72	mov rbp, rax
72	.L7:	
73	74	lea rdi, [rsp+32]
74	75	call std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::basic_string(rdi)
75	76	jmp .L8
76	77	mov rbp, rax
77	.L8:	
78	79	mov rdi, rsp
79	80	call std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::basic_string(rdi)
80	81	mov rdi, rbp
81		call __Unwind_Resume

C Output (0/0) x86-64 gcc (trunk) - 1448ms (212276B)

Returning values

All temporary objects are destroyed as the last step in evaluating the full-expression that (lexically) contains the point where they were created, and if multiple temporary objects were created, they are destroyed in the order opposite to the order of creation.



Testing strings

```
template <typename InputIt, typename T>
T accumulate(InputIt first, InputIt last, T init)
{
    while (first != last) {
        init = init + *first;
        ++first;
    }
    return init;
}
```



```
1 #include <string>
2 #include <vector>
3
4 template<class InputIt, class T, class F>
5 T accumulate(InputIt first, InputIt last, T init, F op)
6 {
7     for ( ; first != last; ++first) {
8         init = op(init, *first);
9     }
10    return init;
11 }
12
13 void f(std::vector<std::string> xs)
14 {
15     accumulate(
16         cbegin(xs), cend(xs), std::string{},
17         [] (std::string acc, const std::string& x)
18             -> std::string
19         {
20             return acc + x;
21         }
22     );
23 }
```

A □ 11010 ☑ .LX0: ☐ lib.f: ☑ .text ☑ // ☑ \s+ ☑ Intel ☑ Demangle

Libraries ▾ + Add new... ▾ ⚙ Add tool... ▾

```
209 mov rax, QWORD PTR [rsp+112]
210 call std::__throw_logic_error(char const*)
211 mov rbx, rax
212 jmp .L14
213 mov rbx, rax
214 jmp .L30
215 mov rbx, rax
216 jmp .L16
217 f(std::vector<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>)
218 .L14:
219     mov rdi, QWORD PTR [rsp+64]
220     lea rax, [rsp+80]
221     cmp rdi, rax
222     je .L16
223     call operator delete(void*)
224 .L16:
225     mov rdi, QWORD PTR [rsp+96]
226     lea rdx, [rsp+112]
227     cmp rdi, rdx
228     je .L30
229     call operator delete(void*)
230 .L30:
231     mov rdi, QWORD PTR [rsp+32]
232     lea rdx, [rsp+48]
233     cmp rdi, rdx
234     je .L32
235     call operator delete(void*)
236 .L32:
237     mov rdi, rbx
238     call _Unwind_Resume
```

C Output (0/0) x86-64 gcc 8.3 - 1157ms (343937B)

```

1 #include <string>
2 #include <vector>
3
4 template<class InputIt, class T, class F>
5 T accumulate(InputIt first, InputIt last, T init, F op)
6 {
7     for ( ; first != last; ++first) {
8         init = op(std::move(init), *first);
9     }
10    return init;
11 }
12
13 void f(std::vector<std::string> xs)
14 {
15     accumulate(
16         cbegin(xs), cend(xs), std::string{},
17         [] (std::string &&acc, const std::string& x)
18             -> std::string
19         {
20             return std::move(acc) + x;
21         }
22     );
23 }

```

A □ 11010 ☑ .LX0: ☑ lib.f: ☑ .text ☑ // ☑ ls+ ☑ Intel ☑ Demangle

Libraries ▾ + Add new... ▾ ⚙ Add tool... ▾

112	call	memcpy
113	mov	rdx, QWORD PTR [rsp+56]
114	mov	rdi, QWORD PTR [rsp+16]
115 .L7:		
116	mov	QWORD PTR [rsp+24], rdx
117	mov	BYTE PTR [rdi+rdx], 0
118	mov	rdi, QWORD PTR [rsp+48]
119	jmp	.L9
120 .L32:		
121	movzx	eax, BYTE PTR [rsp+64]
122	mov	BYTE PTR [rdi], al
123	mov	rdx, QWORD PTR [rsp+56]
124	mov	rdi, QWORD PTR [rsp+16]
125	mov	QWORD PTR [rsp+24], rdx
126	mov	BYTE PTR [rdi+rdx], 0
127	mov	rdi, QWORD PTR [rsp+48]
128	jmp	.L9
129	mov	rbx, rax
130	jmp	.L18
131 f(std::vector<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>> xs) 132 .L18:		
133	mov	rdi, QWORD PTR [rsp+16]
134	lea	rdx, [rsp+32]
135	cmp	rdi, rdx
136	je	.L19
137	call	operator delete(void*)
138 .L19:		
139	mov	rdi, rbx
140	call	_Unwind_Resume

C Output (0/0) x86-64 gcc 8.3 - 939ms (306917B)

```

1 #include <string>
2 #include <vector>
3
4 template<class InputIt, class T, class F>
5 T accumulate(InputIt first, InputIt last, T init, F op)
6 {
7     for (; first != last; ++first) {
8         init = op(std::move(init), *first); // std::move
9     }
10    return init;
11}
12
13 void f(std::vector<std::string> xs)
14 {
15     accumulate(
16         cbegin(xs), cend(xs), std::string{},
17         [] (std::string &&acc, const std::string& x)
18             -> std::string&&
19         {
20             return std::move(acc) + x;
21         }
22     );
23 }

```

If STL used the rvalue
return approach

A □ 11010 ☑ .LX0: ☑ lib.f: ☑ .text ☑ // ☑ \s+ ☑ Intel ☑ Demangle

Libraries ▾ + Add new... ▾ ⚙ Add tool... ▾

29	mov	rdi, QWORD PTR [rsp+32]
30	mov	QWORD PTR [rax+8], 0
31	lea	rax, [rsp+48]
32	cmp	rdi, rax
33	je	.L5
34	call	operator delete(void*)
35		.L5:
36	mov	rax, QWORD PTR ds:0
37	ud2	
38		.L11:
39	movdqu	xmm0, XMMWORD PTR [rax+16]
40	movaps	XMMWORD PTR [rsp+48], xmm0
41	jmp	.L4
42		.L1:
43	add	rsp, 64
44	pop	rbx
45	ret	
46	mov	rbx, rax
47	jmp	.L6
48		f(std::vector<std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>> xs)
49		.L6:
50	mov	rdi, QWORD PTR [rsp]
51	lea	rdx, [rsp+16]
52	cmp	rdi, rdx
53	je	.L7
54	call	operator delete(void*)
55		.L7:
56	mov	rdi, rbx
57	call	_Unwind_Resume

C Output (0/4) x86-64 gcc 8.3 -cached (280850B)

```
1 #include <string>
2 #include <vector>
3
4 template<class InputIt, class T, class F>
5 T accumulate(InputIt first, InputIt last, T init, F op)
6 {
7     for (; first != last; ++first) {
8         init = op(std::move(init), *first); // std::move
9     }
10    return init;
11 }
12
13 void f(std::vector<std::string> xs)
14 {
15     accumulate(
16         cbegin(xs), cend(xs), std::string{},
17         [] (std::string &&acc, const std::string& x)
18             -> std::string&&
19         []
20             {
21                 acc.append(x);
22             }
23     );
24 }
```

A □ 11010 ☑ .LX0: ☐ lib.f: ☑ .text ☑ // ☐ \s+ ☑ Intel ☑ Demangle

Libraries ▾ + Add new... ☰ Add tool... ▾

17	mov	rsi, rsp
18	mov	rdi, rsp
19	add	rbx, 32
20	call	std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::operator=(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>)
21	jmp	.L3
22	.L2:	
23	mov	rsi, rsp
24	lea	[rsp+32]
25	call	std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::operator=(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>)
26	lea	[rsp+32]
27	call	std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::operator=(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>)
28	mov	rdi, rsp
29	call	std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::operator=(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>)
30	add	rsp, 72
31	pop	rbx
32	pop	rbp
33	ret	
34	mov	rbx, rax
35	mov	rdi, rsp
36	call	std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>::operator=(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>)
37	mov	rdi, rbx
38	call	_Unwind_Resume

C Output (0/4) x86-64 gcc 8.3 -cached (280850B)

Testing strings

- Consider returning &&
- But be cautious of dangling references
- Store result by-value



Testing strings

```
for (auto x: foo().value() ) {  
}
```



Testing strings

```
for (auto f = foo(); auto x: f.value()) {  
}
```



LINEAR IN C++

Linear in C++

- Moving is required
- Copies should be disallowed
- Moves should be efficient (*)



Far away worlds
ooo

Attack of the Clones
oooooooooooooo

Generic
oooooooooooooooo

Performance
oooooooooooooooo

Linear in C++
oo●oooooooooooo

The End
o

Moving

- T can be *seen* as T
- T&& can be *seen* as T



Moving

`detail::linear_usable_as_v<T, T>` and
`detail::linear_usable_as_v<T, T&&>`



Moving

```
namespace detail {

    template <typename T, typename U>
    constexpr bool linear_usable_v =

        std::is_nothrow_constructible_v<T, U> and
        std::is_nothrow_assignable_v<T&, U> and
        std::is_nothrow_convertible_v<U, T>;

}
```



No copies allowed

- T& is not T
- const T& is not T
- const T is not T



Far away worlds
ooo

Attack of the Clones
oooooooooooooo

Generic
oooooooooooooooooooo

Performance
oooooooooooooooooooo

Linear in C++
ooooo●oooooooooooo

The End
○

Gray place

There's a thin line between love and hate
Wider divide that you can see between good and
bad
There's a grey place between black and white

Dave Murray, Steve Harris



No copies allowed

`detail::linear_unusable_as_v<T, T&>` and
`detail::linear_unusable_as_v<T, const T&>` and
`detail::linear_unusable_as_v<T, const T>`

No copies allowed

```
namespace detail {

    template <typename T, typename U>
    constexpr bool linear_unusable_as_v =

        not std::is_constructible_v<T, U> and
        not std::is_assignable_v<T&, U> and
        not std::is_convertible_v<U, T>;

}
```



Linear in C++

```
template <typename T>
concept Linear =
    std::is_nothrow_destructible_v<T> and

    detail::linear_usable_as<T, T> and
    detail::linear_usable_as<T, T&&> and

    detail::linear_unusable_as<T, T&> and
    detail::linear_unusable_as<T, const T&> and
    detail::linear_unusable_as<T, const T>;
```

Linear in C++

```
auto ptr = std::make_unique<person>();  
  
auto str = "Hello, Italian C++!"s;
```



Linear in C++

```
Linear ptr = std::make_unique<person>(); // OK  
  
Linear str = "Hello, Italian C++!"s; // ERROR
```



Linear in C++

```
template <typename T>
    requires(Linear<T>)
auto accumulate(auto xs, T init)
{
    ...
}
```



Linear in C++

```
template <Linear T>
auto accumulate(auto xs, T init)
{
    ...
}
```



Linear in C++

```
auto accumulate(auto xs, Linear auto init)
{
    ...
}
```

Wrapper

What to do with non-linear types?

Linear wrapper

```
template <typename T>
class linear_wrapper {
public:
    linear(const linear&) = delete;
    linear(linear&&) = default; // noexcept

    linear& operator=(const linear&) = delete;
    linear& operator=(linear&&) = default; // noexcept

    ...
private:
    T m_value;
};
```



Linear wrapper

```
template <typename T>
class linear_wrapper {
public:
    linear_wrapper(T&& value)
        : m_value{std::move(value)}
    {
    }
    ...
private:
    T m_value;
};
```

rvalue ref. -- so
we use move on it

Linear wrapper

```
template <typename T>
class linear_wrapper {
public:
    template <typename... Args>
    linear_wrapper(std::in_place_t, Args&&... args)
        : m_value(std::forward<Args>(args)...)
    {
    }

    ...

private:
    T m_value;
};
```



Linear wrapper

```
template <typename T>
class linear_wrapper {
public:
    [[nodiscard]] T&& get() && noexcept
    {
        return std::move(value);
    }

    ...

private:
    T m_value;
};
```

Linear wrapper

```
template <typename T>
class linear_wrapper {
public:
    [[nodiscard]] T&& operator*() && noexcept
    {
        return std::move(value);
    }

    ...

private:
    T m_value;
};
```

Linear wrapper

```
auto operator""_ls(const char* data,  
                   std::size_t len)  
{  
    return linear_wrapper<std::string>(std::in_place, data);  
}  
  
accumulate(in, "Concatenated:_ls); // ERROR before C++20
```

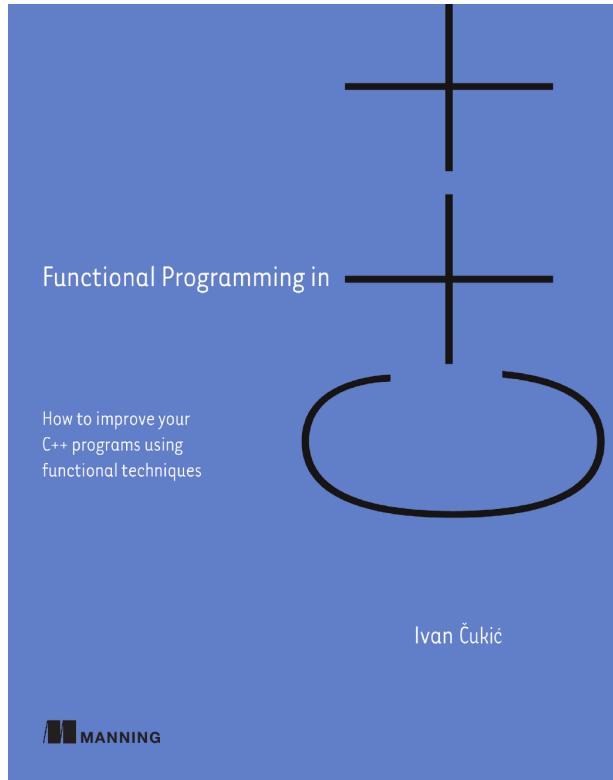


Additional

- Use after move
(clang-tidy:bugprone-use-after-move)
- Unused variable error
(-Werror=unused-variable)
- Error handling
(optional<T>, expected<T,E>)



Answers? Questions! Questions? Answers!



cukic.co/to/fp-in-cpp
Functional Programming in C++





Phil Nash

开发者测试专家，Catch创始人

Phil Nash 是著名的单元测试工具Catch/Catch2的创始人，作为 JetBrains 的技术专家，他的工作主要涉及CLion, AppCode 和 ReSharper C++。他是优秀测试实践、测试驱动开发 (TDD) 的倡导者。他曾长期担任技术顾问和技术教练工作。他同时C++ London 的创办人，C++ on Sea的主席。

主办方：

Boolan
高端IT咨询与教育平台



Test First Test Better

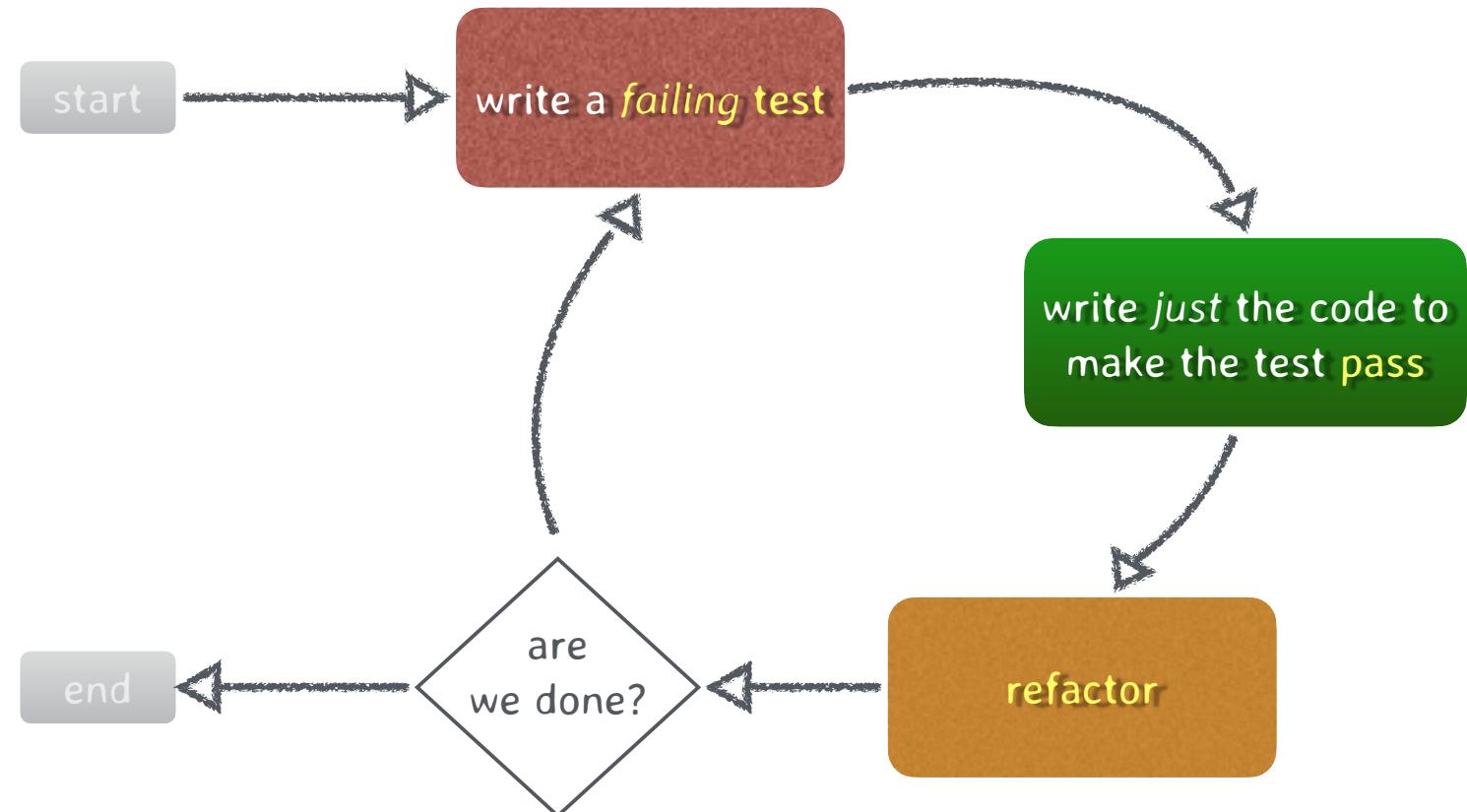
@phil_nash

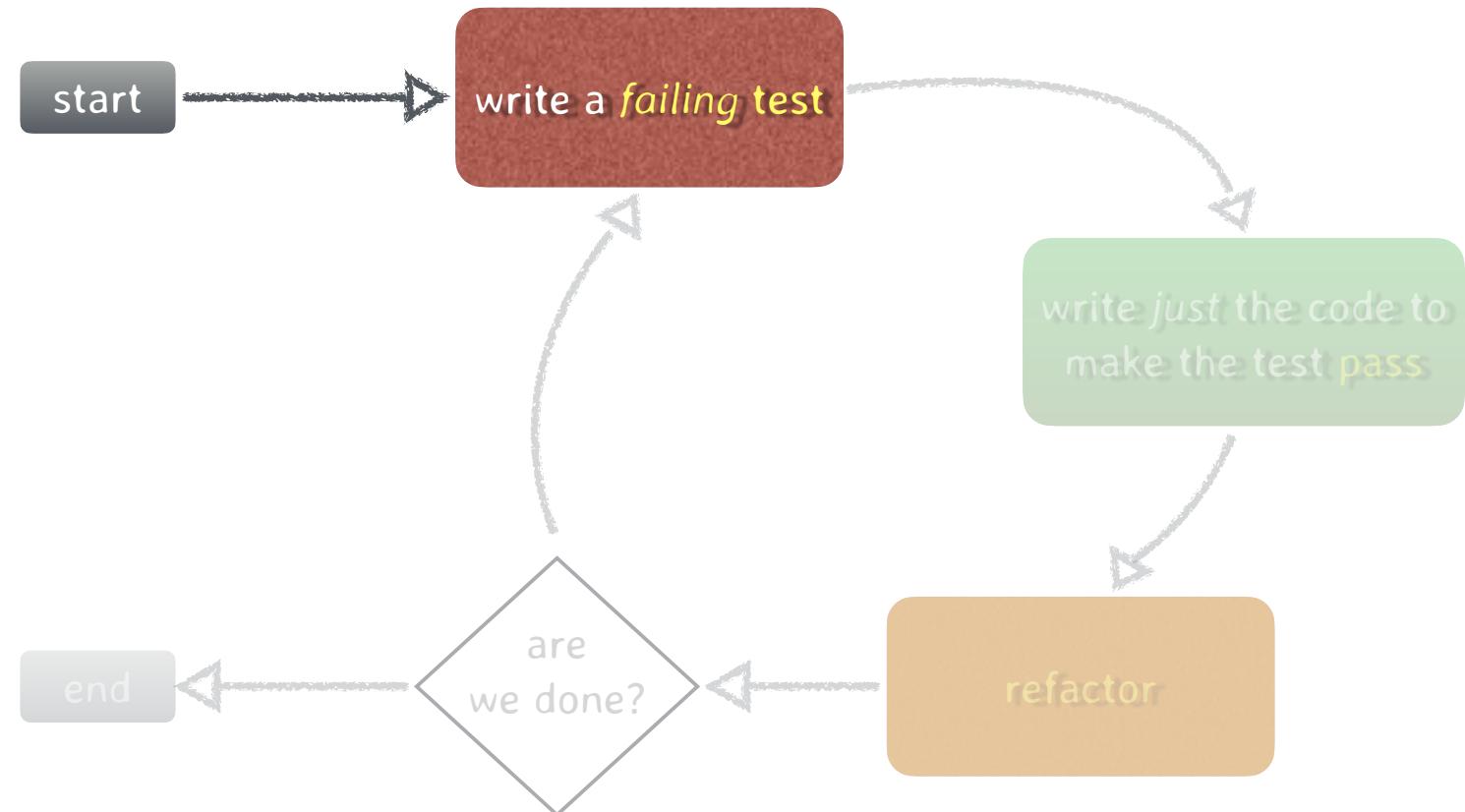
TDD

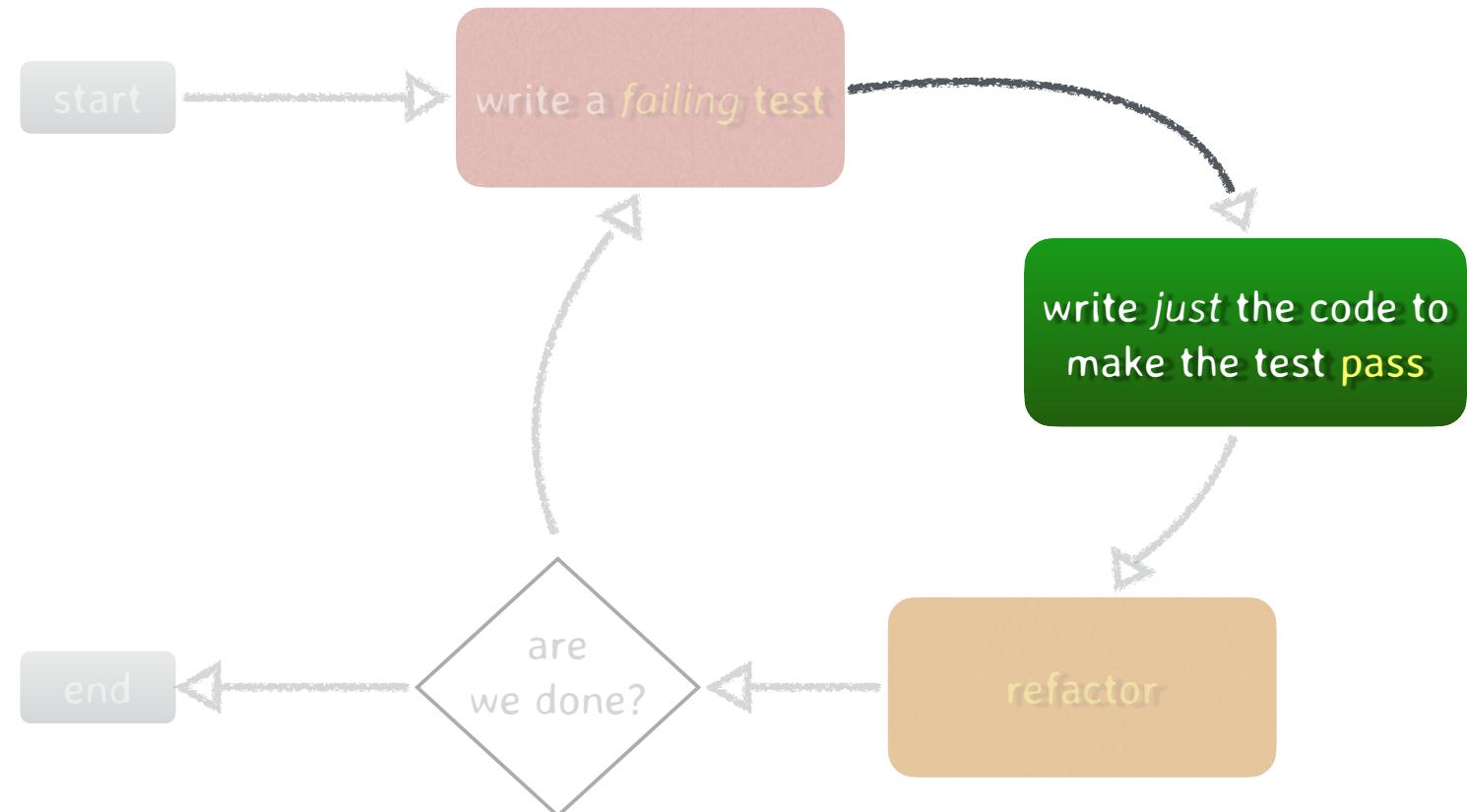
What is TDD?

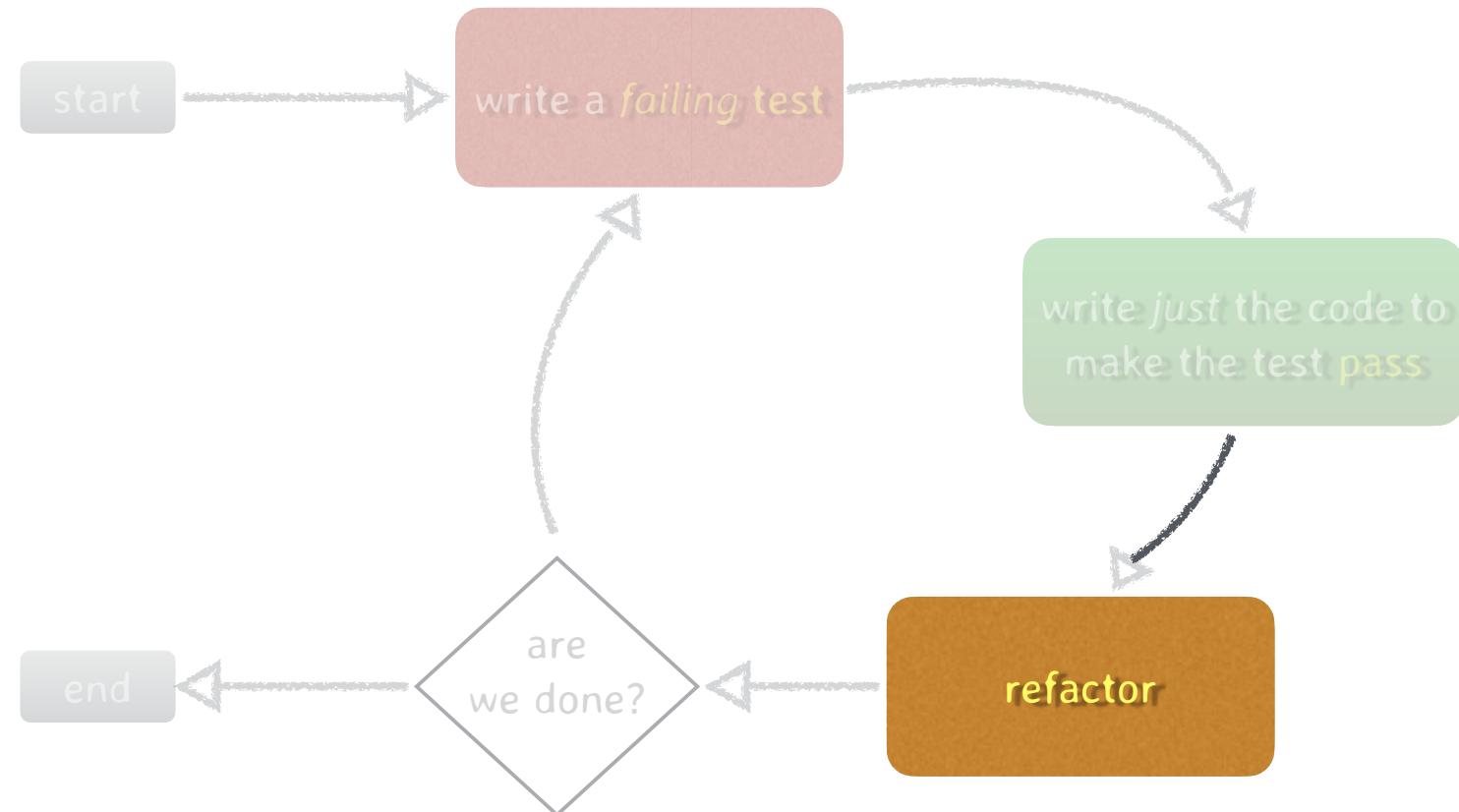
Test Driven Development

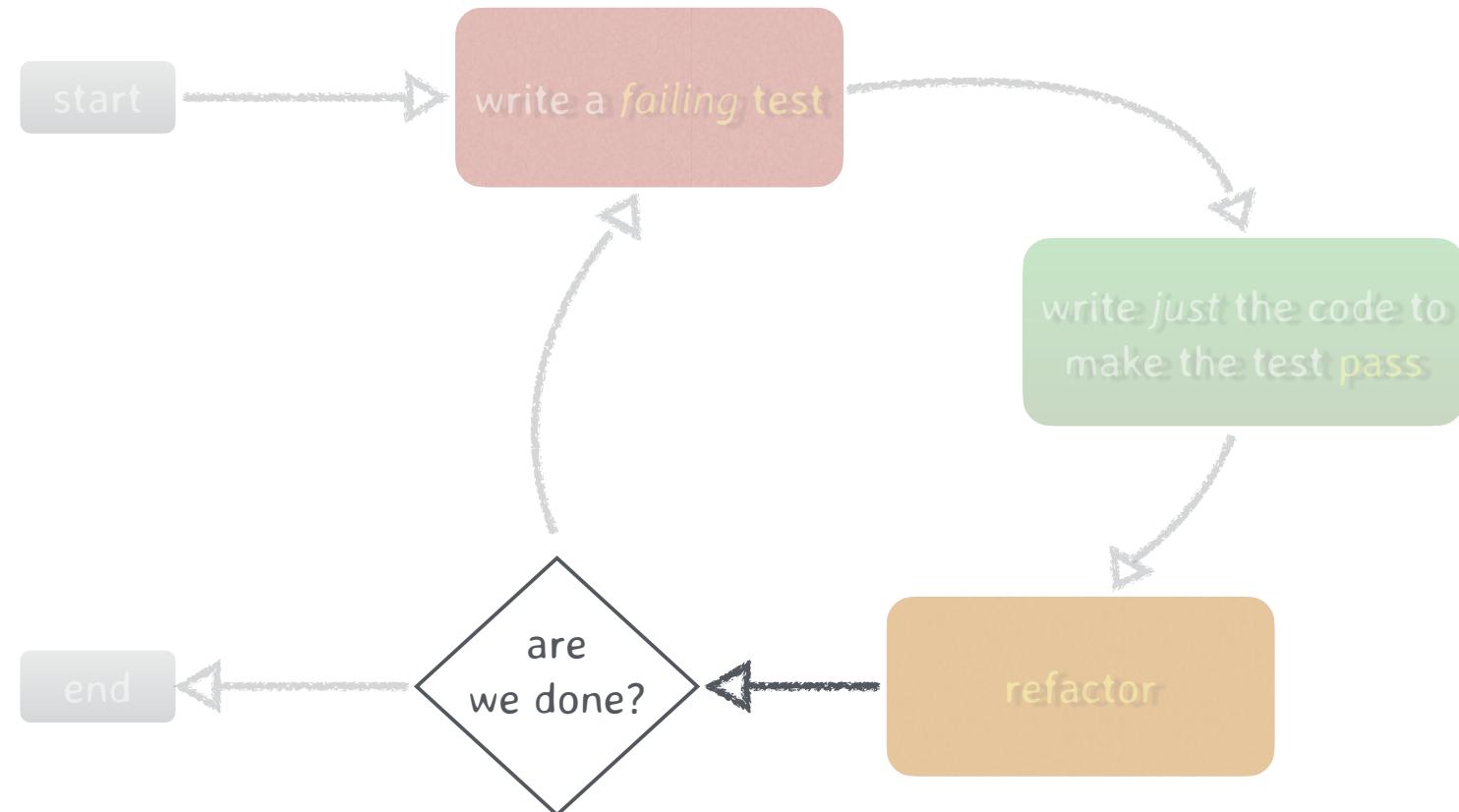
Test Driven *Design*

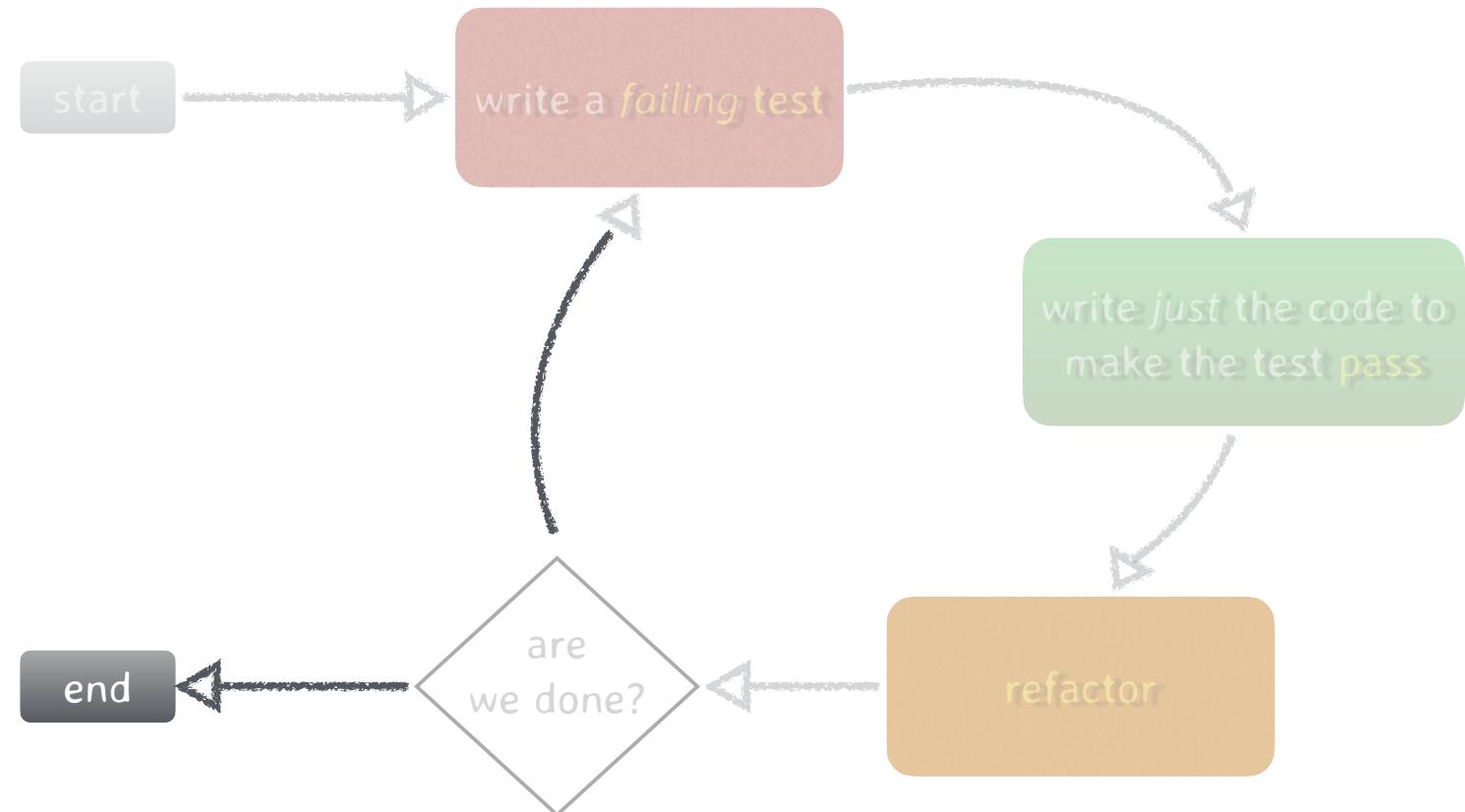






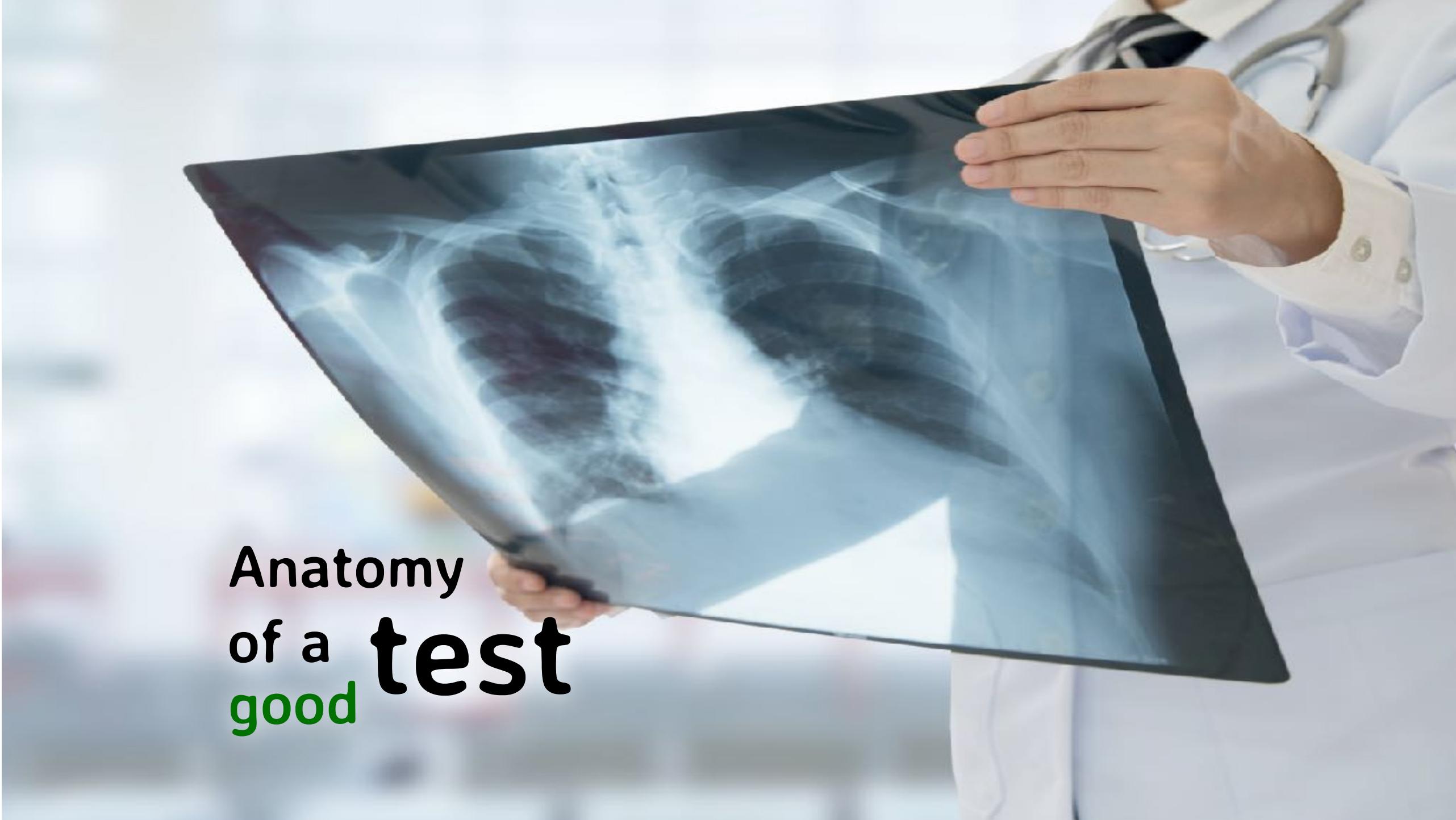








Questions?

A close-up photograph of a doctor's hands holding a chest X-ray film. The doctor is wearing a white medical coat and a stethoscope around their neck. The X-ray film shows the internal structures of the lungs and ribcage. The doctor's hands are positioned to hold the corners of the film.

Anatomy
of a **test**
good

```
TEST_CASE("add() returns the sum of its arguments") {  
    REQUIRE( add( 1, 2 ) == 3 );  
}
```

```
TEST_CASE("add() returns the sum of its arguments") {  
    REQUIRE( add( 1, 2 ) == 3 );  
}
```

TESTS SHOULD HAVE A
GOOD NAME

```
TEST_CASE("add() returns the sum of its arguments") {  
    REQUIRE( add( 1, 2 ) == 3 );  
}
```

```
TEST_CASE( "Most recently used list" ) {  
    MRULList<std::string> list;  
  
    SECTION( "An empty list has no elements" ) {  
        REQUIRE( list.empty() );  
        REQUIRE( list.size() == 0 );  
    }  
    SECTION( "Adding to an empty list increases the size to 1" ) {  
        list.add("item1");  
        REQUIRE( list.empty() == false );  
        REQUIRE( list.size() == 1 );  
    }  
}
```

```
TEST_CASE( "An MRU list acts like a stack, "
           "but duplicate entries replace existing ones" ) {

    MRULList<std::string> list;

    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }
    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

```
TEST_CASE( "An MRU list acts like a stack, "
           "but duplicate entries replace existing ones" ) {

    MRULList<std::string> list;

    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }

    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

```
TEST_CASE( "An MRU list acts like a stack, "
           "but duplicate entries replace existing ones" ) {

    MRULList<std::string> list;

    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }

    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

STATE EXPECTATIONS

UNIT TESTS SHOULD HAVE A REGULAR STRUCTURE

```
TEST_CASE( "An MRU list acts like a stack, "
           "but duplicate entries replace existing ones" ) {

    MRUList<std::string> list;

    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }

    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

UNIT TESTS SHOULD HAVE A REGULAR STRUCTURE

```
TEST_CASE( "An MRU list acts like a stack, "
           "but duplicate entries replace existing ones" ) {
    MRUList<std::string> list;
    "ARRANGE"
    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }
    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

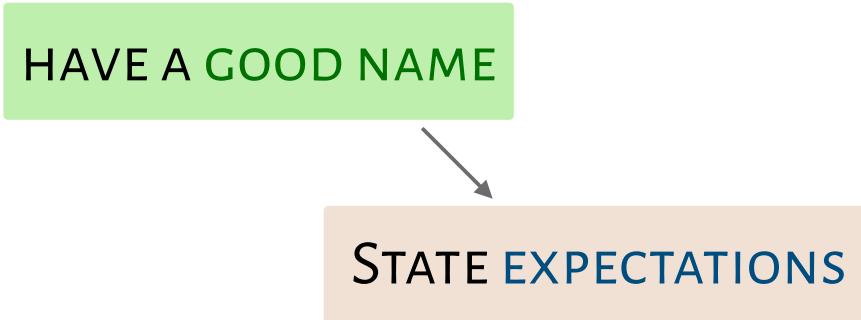
UNIT TESTS SHOULD HAVE A REGULAR STRUCTURE

```
TEST_CASE( "An MRU list acts like a stack, "
           "but duplicate entries replace existing ones" ) {
    MRUList<std::string> list;
    "ARRANGE"
    SECTION( "An empty list has no elements" ) {
        REQUIRE( list.empty() );
        REQUIRE( list.size() == 0 );
    }
    SECTION( "Adding to an empty list increases the size to 1" ) {
        list.add("item1");
        "ACT"
        REQUIRE( list.empty() == false );
        REQUIRE( list.size() == 1 );
    }
}
```

UNIT TESTS SHOULD HAVE A REGULAR STRUCTURE

```
TEST_CASE( "An MRU list acts like a stack,  
           \"but duplicate entries replace existing ones\" ) {  
  
    MRUList<std::string> list;                                "ARRANGE"  
  
    SECTION( "An empty list has no elements" ) {  
        REQUIRE( list.empty() );  
        REQUIRE( list.size() == 0 );  
    }  
    SECTION( "Adding to an empty list increases the size to 1" ) {  
        list.add("item1");                                         "ACT"  
        REQUIRE( list.empty() == false );  
        REQUIRE( list.size() == 1 );  
    }  
}
```

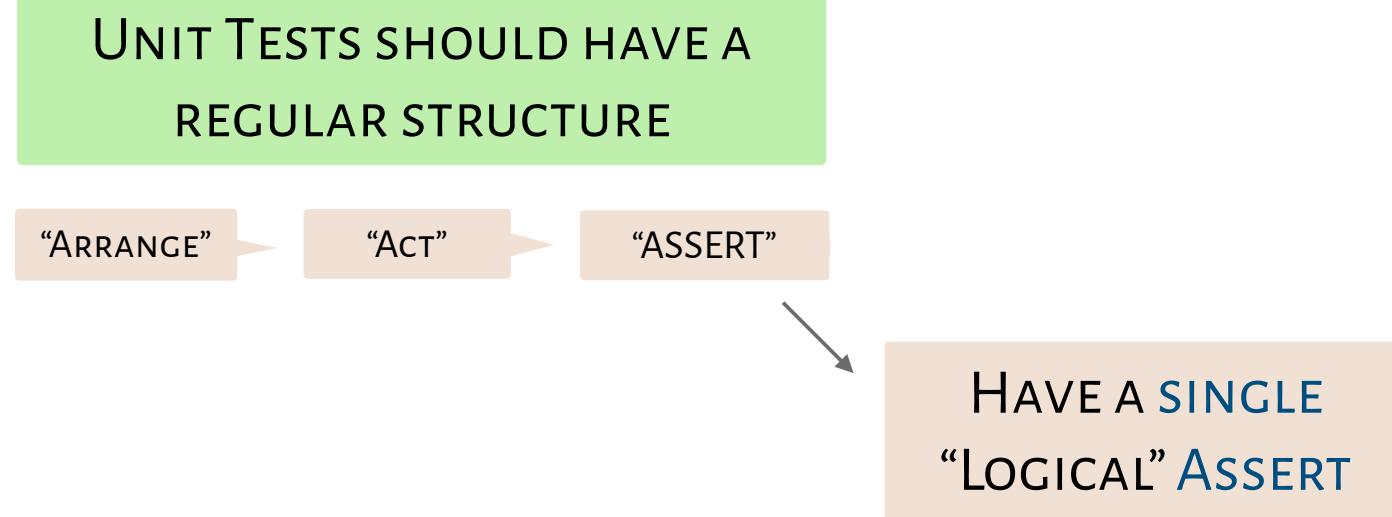
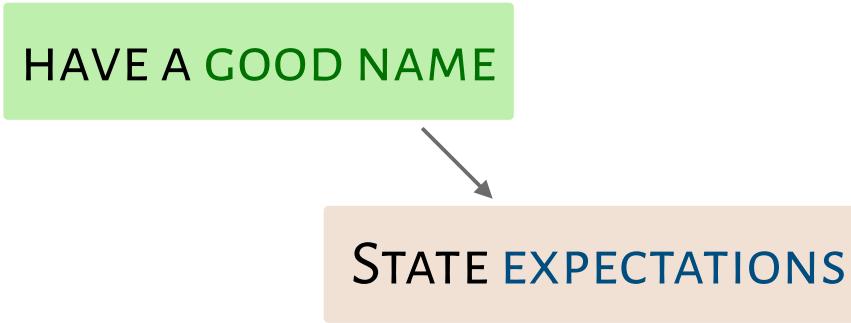
TESTS SHOULD:



UNIT TESTS SHOULD HAVE A
REGULAR STRUCTURE

“ARRANGE” → “Act” → “ASSERT”

TESTS SHOULD:



TESTS SHOULD:

HAVE A **SINGLE**
“LOGICAL” **ASSERT**

```
TEST_CASE( "An MRU list acts like a stack,  
          "but duplicate entries replace existing ones" ) {  
  
    MRUList<std::string> list;  
  
    SECTION( "An empty list has no elements" ) {  
        REQUIRE( list.empty() );  
        REQUIRE( list.size() == 0 );  
    }  
    SECTION( "Adding to an empty list increases the size to 1" ) {  
        list.add("item1");  
        REQUIRE( list.empty() == false );  
        REQUIRE( list.size() == 1 );  
    }  
}
```

SINGLE ASSERT

TESTS SHOULD:

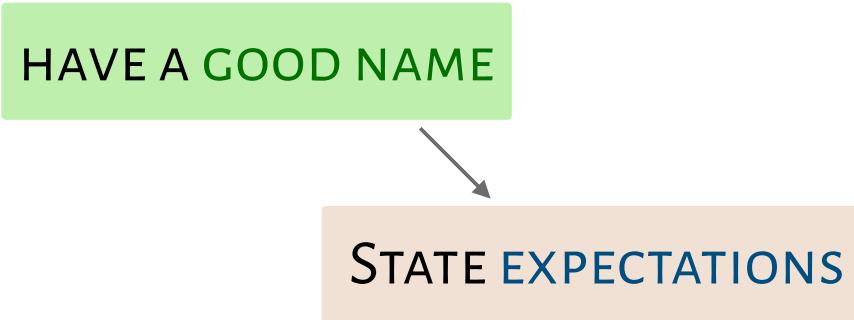
HAVE A SINGLE
“LOGICAL” ASSERT

```
TEST_CASE( "An MRU list acts like a stack,  
          \"but duplicate entries replace existing ones\" ) {  
  
    MRUList<std::string> list;  
  
    SECTION( "...?" ) {  
        REQUIRE( list.empty() );  
        REQUIRE( list.size() == 0 );  
  
        list.add("item1");  
  
        REQUIRE( list.empty() == false );  
        REQUIRE( list.size() == 1 );  
    }  
}
```

DEPENDENCY

MULTIPLE ASSERTS

TESTS SHOULD:



UNIT TESTS SHOULD HAVE A
REGULAR STRUCTURE

“ARRANGE” → “Act” → “ASSERT”

```
graph TD; A[“ARRANGE”] --> B[“Act”]; B --> C[“ASSERT”]; C --> D[HAVE A SINGLE LOGICAL ASSERT]
```

A flowchart with four boxes. The first three boxes are orange and contain the words "ARRANGE", "Act", and "ASSERT" respectively, arranged horizontally with arrows between them. An arrow points from the "ASSERT" box down to the fourth box, which is light orange and contains the text "HAVE A SINGLE LOGICAL ASSERT".

TESTS SHOULD:

HAVE A GOOD NAME

STATE EXPECTATIONS

USE PUBLIC INTERFACE

UNIT TESTS SHOULD HAVE A
REGULAR STRUCTURE

“ARRANGE”

“Act”

“ASSERT”

HAVE A SINGLE
“LOGICAL” ASSERT

TESTS SHOULD: USE PUBLIC INTERFACE

Q. How do you test private methods?

TESTS SHOULD: USE PUBLIC INTERFACE

Q. How do you test private methods?

A. You don't

TESTS SHOULD: USE PUBLIC INTERFACE

Q. How do you test private methods?

A. You don't

Q. But what if you really need to?

TESTS SHOULD: USE PUBLIC INTERFACE

```
friend class TestAccess;
```

TESTS SHOULD: USE PUBLIC INTERFACE

```
friend class TestAccess;
```

To test “non-functional” requirements
(ref counts, caches etc)

TESTS SHOULD: USE PUBLIC INTERFACE

```
friend class TestAccess;
```

```
extern name_not_in_header;
```

TESTS SHOULD: USE PUBLIC INTERFACE

```
friend class TestAccess;
```

```
extern name_not_in_header;
```

```
#define private public;
```

TESTS SHOULD:

HAVE A GOOD NAME

STATE EXPECTATIONS

USE PUBLIC INTERFACE

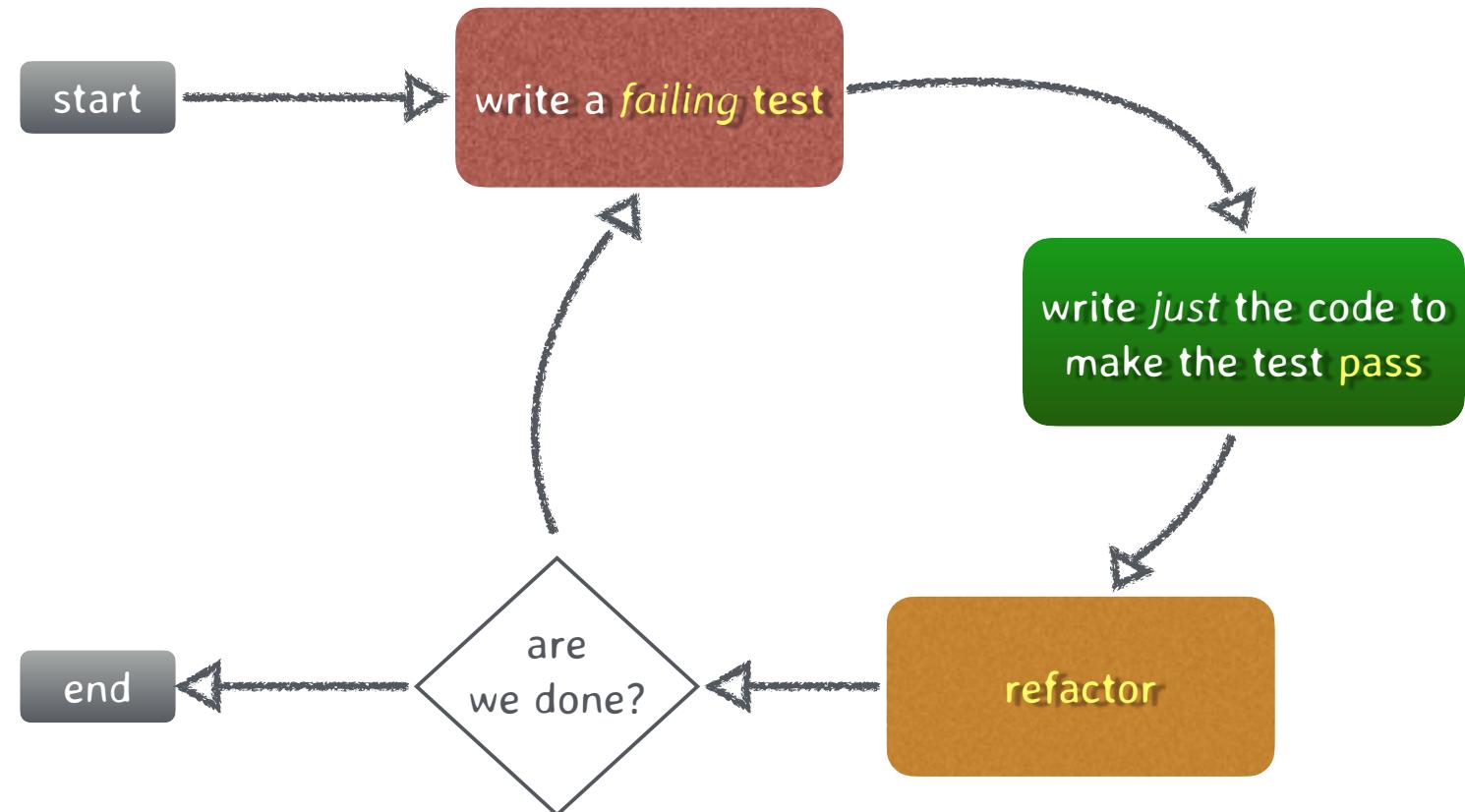
UNIT TESTS SHOULD HAVE A
REGULAR STRUCTURE

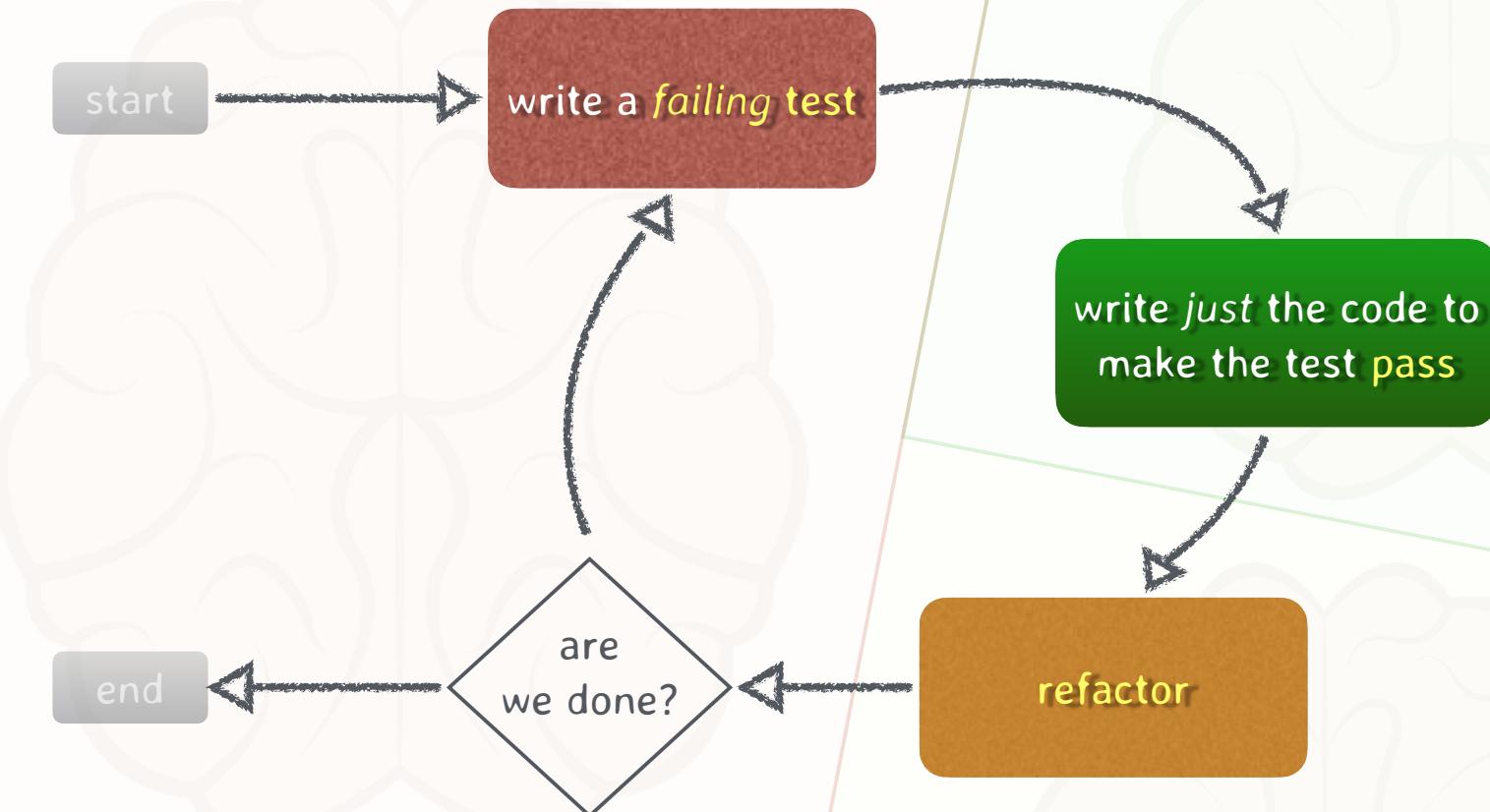
“ARRANGE”

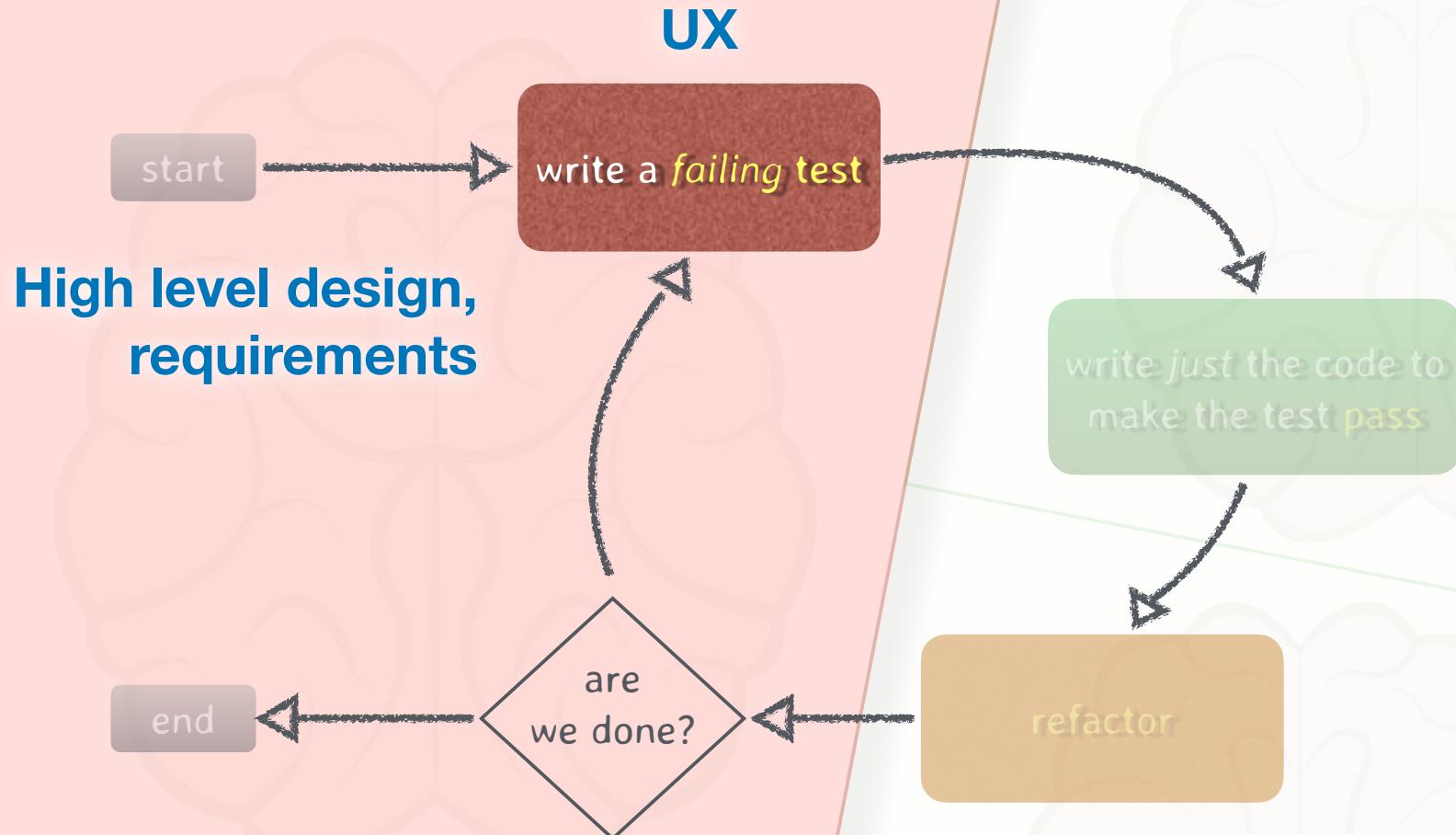
“Act”

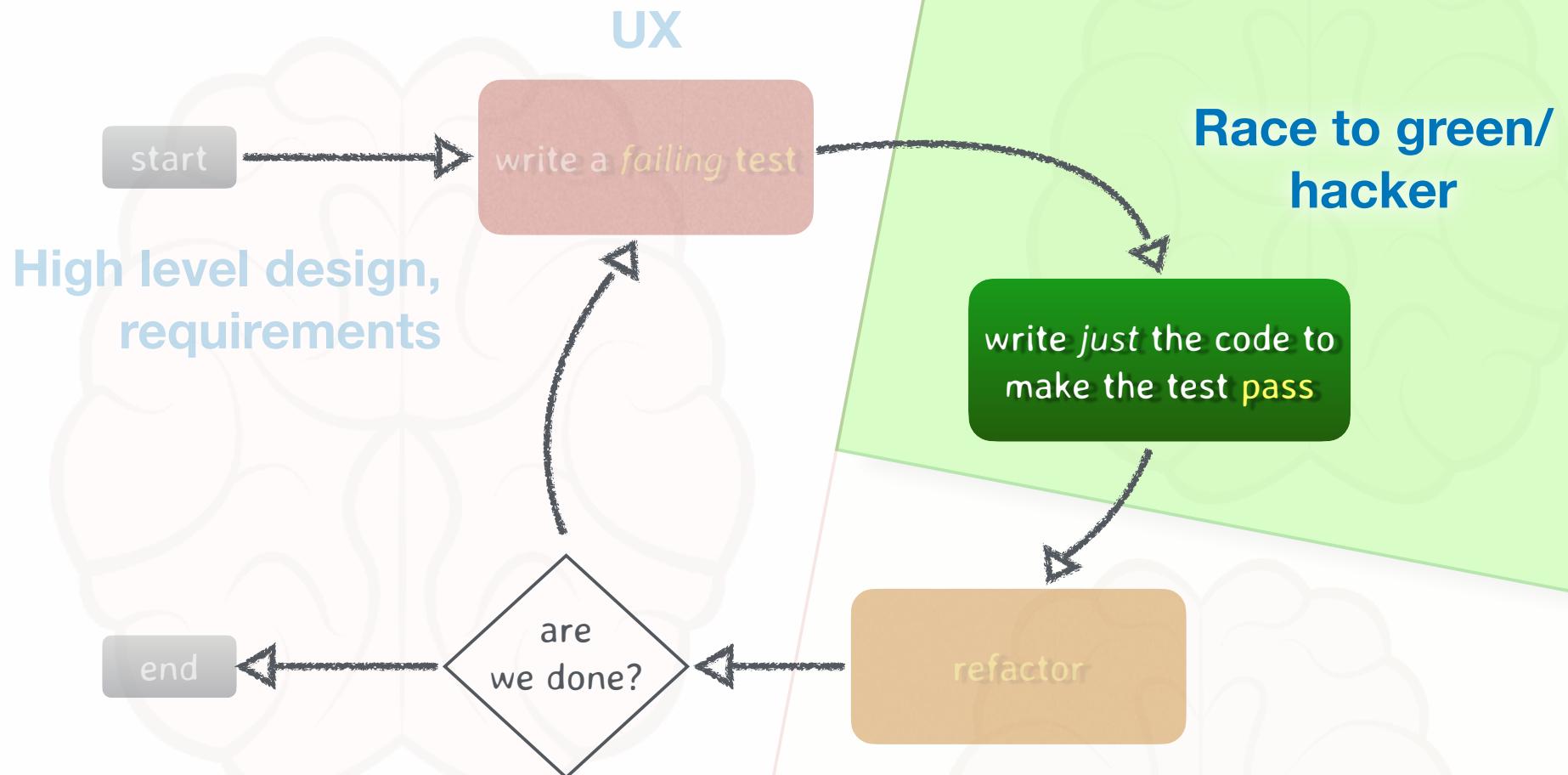
“ASSERT”

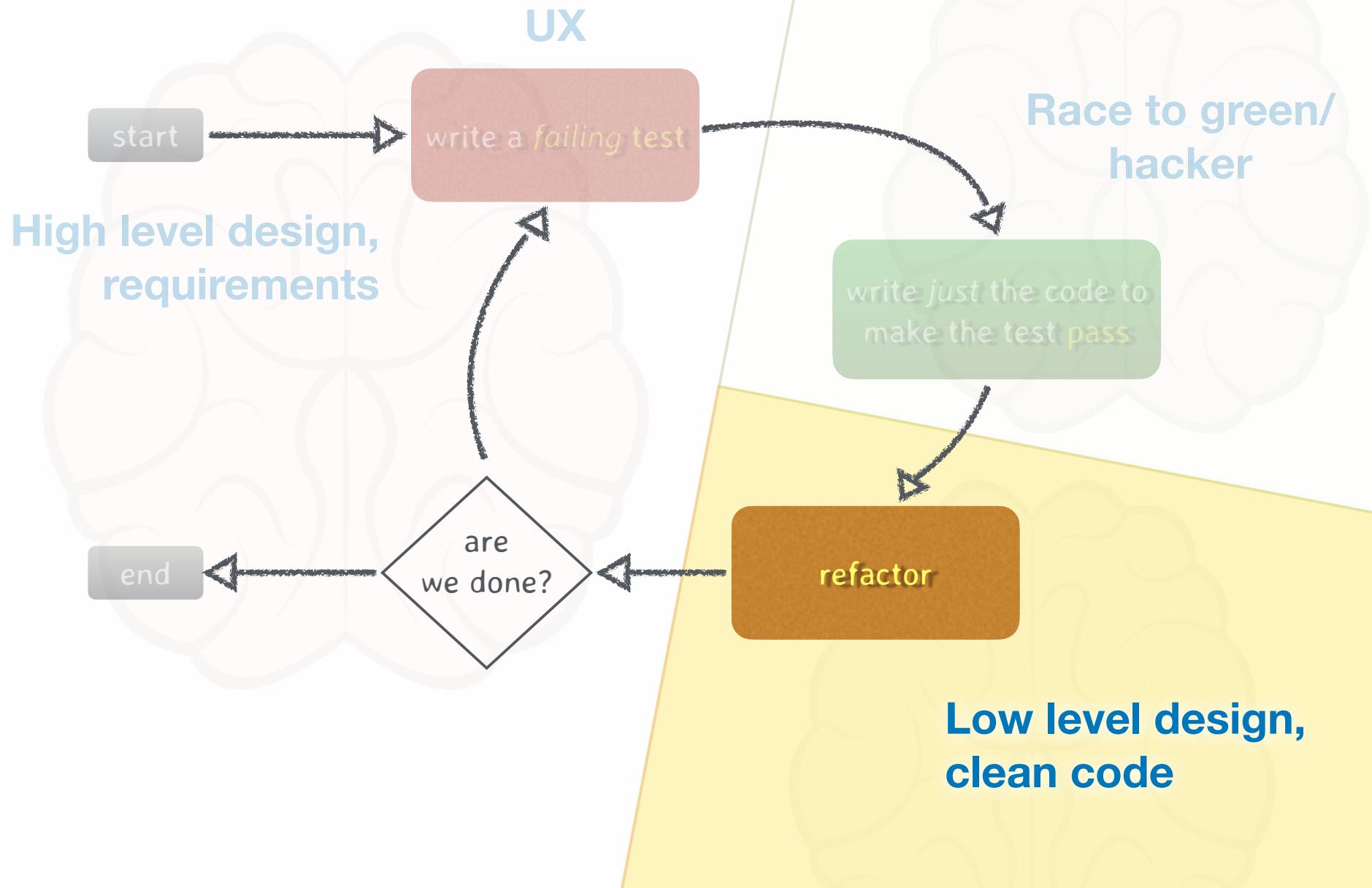
HAVE A SINGLE
“LOGICAL” ASSERT

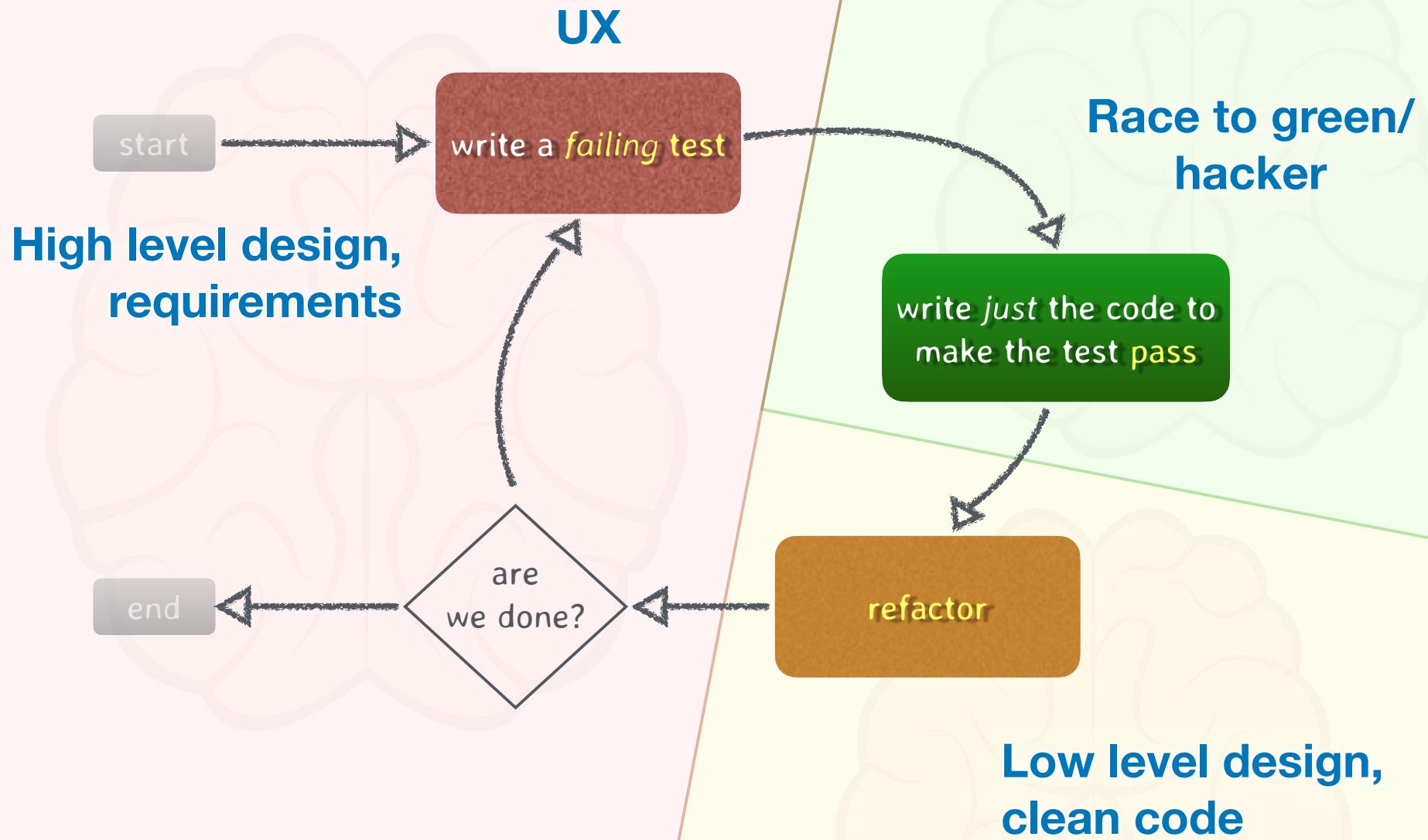












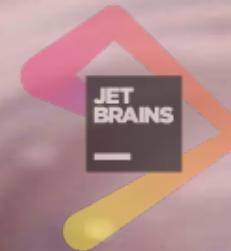
The background image shows a park at sunset. In the foreground, there are several round wooden tables and chairs. A few people are sitting at one of the tables on the left side. The sky is a gradient from blue to orange and yellow. In the distance, there are mountains and a city skyline. The overall atmosphere is peaceful and scenic.

Property Based Testing

Mutation Testing

Legacy Code

Design Principles



Test First Test Better

@phil_nash



陈峰

腾讯广告工程效能负责人，腾讯开源构建系统 Blade 负责人

腾讯开源构建系统 blade 的负责人和核心开发者。拥有二十多年的 C++ 使用经验。负责腾讯广告的代码库管理，基础库建设，构建系统，服务框架开发，DevOps 平台建设，研发文化改进等工作。

主办方：

Boolan
高端 IT 咨询与教育平台

C++ Summit 2020

陈峰

腾讯广告效能负责人

高路

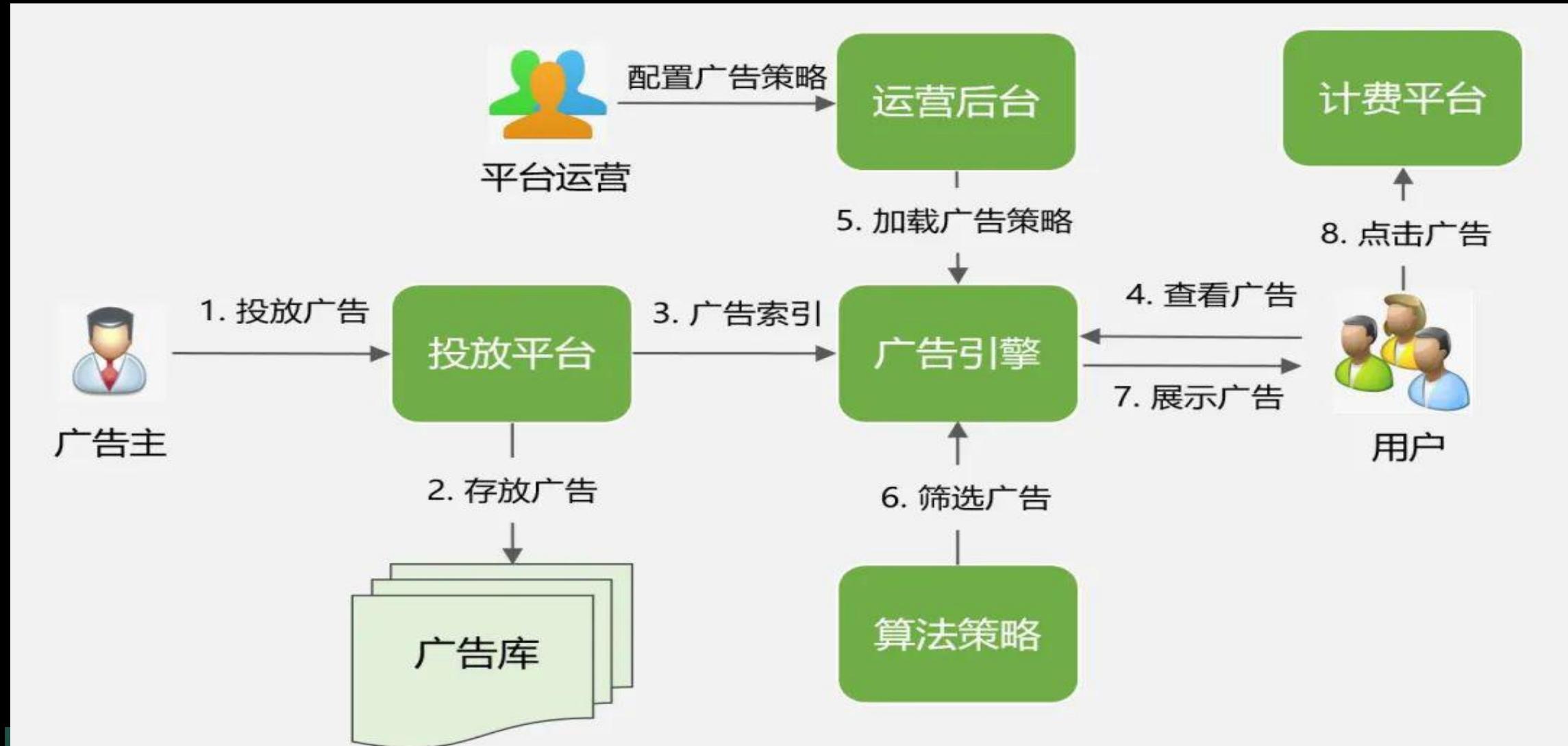
腾讯广告高级工程师

腾讯广告 大规模C++工程实践

什么是腾讯广告

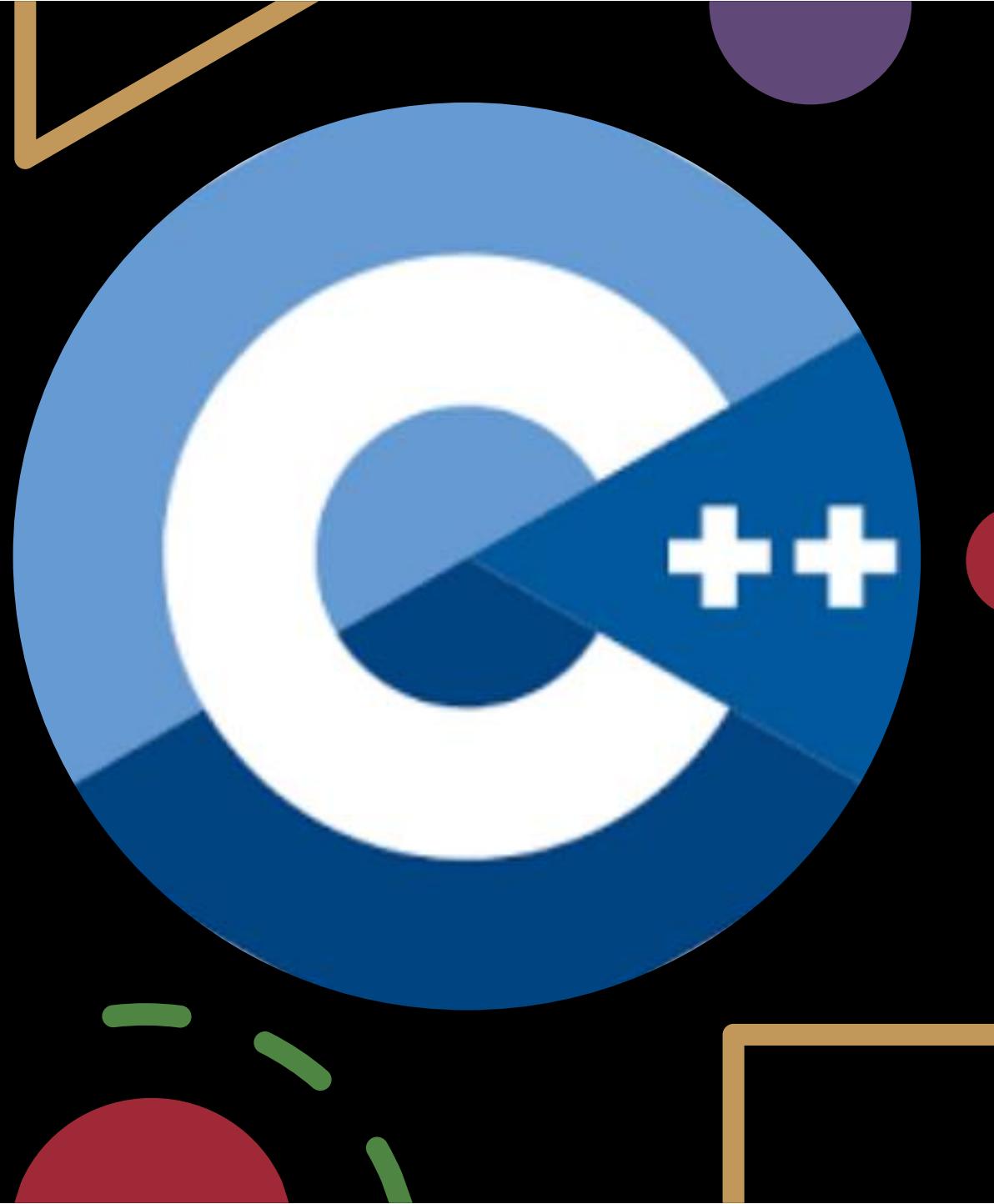


广告系统基本架构



议题

- 代码库管理
- 平台工具建设
- 现代C++实践



代码库管理

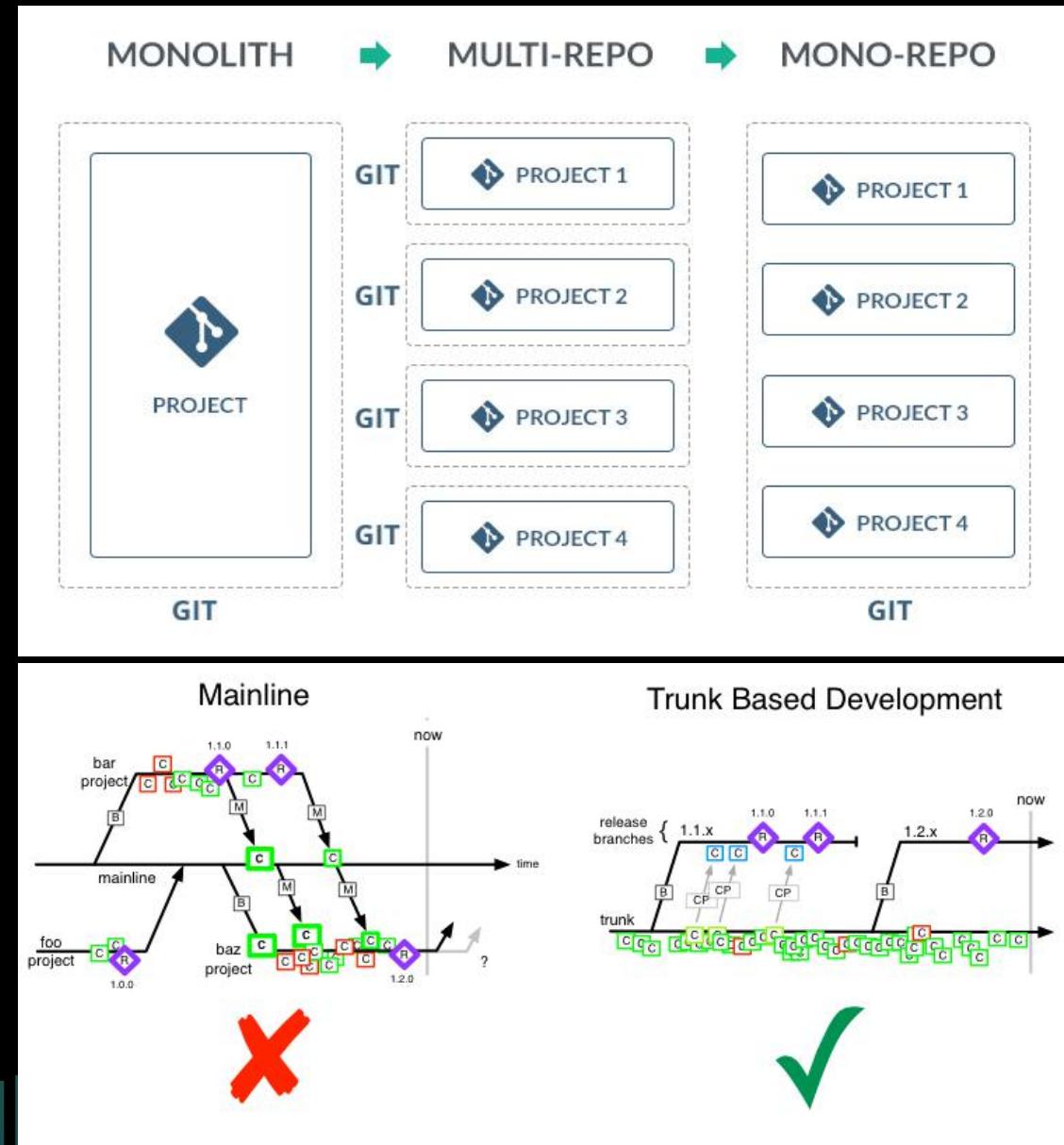
代码库管理——统一组织方式和代码规范

- 基于Google C++代码规范
- 制定《代码组织规范》
- 通过工具检查和代码评审来实施



代码库管理——单一代码库

- 绝大多数代码对所有人可以访问
- 有利于倡导代码公有的理念，方便代码复用
- 方便跨语言共享代码
- 让有问题的代码尽早暴露
- 方便实施原子提交
- 方便实施代码重构
- 不用考虑兼容旧版本
- 方便实施统一的工具检查
- 方便实施主干开发模式



代码库管理——基本组织

- 按项目划分目录，并设置评审人，要有README
- 目录名只和项目关联，不和组织架构关联，因为后者容易变动
- 各级目录名都要有含义，不用include, src 等目录名
- 头文件必须自足性，使用时不需要考虑包含顺序
- 每个功能模块的头文件，实现文件和测试代码都放在同一个目录下，方便发现缺少测试的情况

```
> featureflags  
> feeds  
∨ flare  
  > base  
  > doc  
  > example  
  > fiber  
  > http  
  > init  
  > io  
  > rpc  
  > testing  
  > tools  
≡ .clang-format  
≡ BUILD  
⌚ compat.cc  
⌚ compat.h  
⌚ init.cc  
⌚ init.h  
≡ OWNERS  
ⓘ README.md
```

代码库管理——权限管理

- `/common/proto/OWNERS`

```
set norecurse # 不继承父目录设置 (默认为继承)
martin; mark; michael
```

```
# 支持逻辑表达式运算
per-file *.proto ([zhang3;lisi], [wang5,zhao6])
```

```
# 三人中任意两人通过即可
per-file config.json (terry;peter;saren){2}
```

```
CC=alex;john;mike # 抄送, 无评审通过权
```

- `.github/CODEOWNERS`

```
/common/net/ @pony @martin @mark
```

```
/common/net/*.proto ?
```

CC?

代码库管理——统一构建系统

- 声明式语法
- 自动传递依赖关系
- 支持多种编程语言，方便跨语言调用
- 多任务/分布式构建
- 最小化增量构建
- 内置测试支持

```
cc_library(  
    name = 'ip_address',  
    srcs = ['ip_address.cpp'],  
    hdrs = ['ip_address.h'],  
)  
  
cc_library(  
    name = 'socket',  
    srcs = 'socket.cc',  
    hdrs = 'socket.h',  
    deps = ':socket_address'  
)  
  
cc_library(  
    name = 'socket_address',  
    srcs = 'socket_address.cc',  
    hdrs = 'socket_address.h',  
    deps = [  
        ':ip_address',  
        '//common/base/string:string',  
        '//thirdparty/glog:glog',  
    ]  
)
```

代码库组织——头文件用全路径包含

- 传统的非全路径包含
- 全路径包含

```
#include "http_server.h"
#include "rpc_server.h"

#include "gflags.h"
#include "logging.h"

#include "ad_rewriter.h"
#include "auction_log_report.h"
#include "query_rewrite_service_impl.h"
```

```
#include "common/net/http/server/http_server.h"
#include "common/rpc/rpc_server.h"

#include "thirdparty/gflags/gflags.h"
#include "thirdparty/glog/logging.h"

#include "beeble/ad/ad_rewriter.h"
#include "beeble/query/auction_log_report.h"
#include "beeble/query/query_rewrite_service_impl.h"
```

代码库组织——区分公开头文件和私有头文件

- 通过构建系统控制头文件的可见性
- 不依赖某个库，就不能包含其头文件
- 不能包含库的私有头文件

```
cc_library(  
    name = "http",  
    srcs = [  
        "http.cc",  
        "http_impl.h",  
    hdrs = ["http.h"],  
)
```

代码库组织——命名空间

- 为了避免潜在的名字冲突项目必须采用命名空间
- 命名空间的名字基于项目名
- 必要时可以使用嵌套命名空间
- 如果必须用宏，宏也需要有前缀

```
// In mixer/mixer_server.h

namespace gdt::mixer {

    class MixerServer {
        // ...
    };

} // namespace gdt::mixer

#define GDT_MIXER_DEFINE_PROPERTY(...)
```

代码库组织——库也用全路径

```
cc_binary(  
    name = "mixer_server",  
    ...  
    deps = [  
        ":mixer_proto",  
        "//common/base/string:string",  
        "//common/net:ip_address",  
        "//common/rpc:rpc_server",  
    ],  
)
```

```
# 非全路径链接  
LINKFLAGS= \  
    -L. -Imixer_proto \  
    -L common/base/string -Istring \  
    -L common/net -l ip_address \  
    -L common/rpc -l rpc_server
```

代码库组织——控制库的使用范围

- 通过构建系统控制一些库的使用范围
- 业务模块的库也可以控制自己的可见性，避免被别的项目意外使用

```
cc_library(  
    name = 'boost',  
    deps = [  
        '//thirdparty/boost_1_69_0:boost',  
    ],  
    visibility = [  
        '//thirdparty/thrift:thrift',  
        '//thirdparty/click_house:click_house',  
        '//exp/radar:radar',  
    ],  
)
```

代码库组织——统一管理第三方库

- 制定《第三方库管理规范》
- 统一放在一个目录下
- 禁止项目私有第三方库
- 统一版本
- 专人评审、维护、升级

```
thirdparty
|-- boost
|-- curl
|-- flatbuffers
|-- fmt
|-- gflags
|-- glog
|-- googletest
|-- hdfs
|-- jemalloc
|-- jsoncpp
|-- leveldb
```

代码库组织——基础库建设

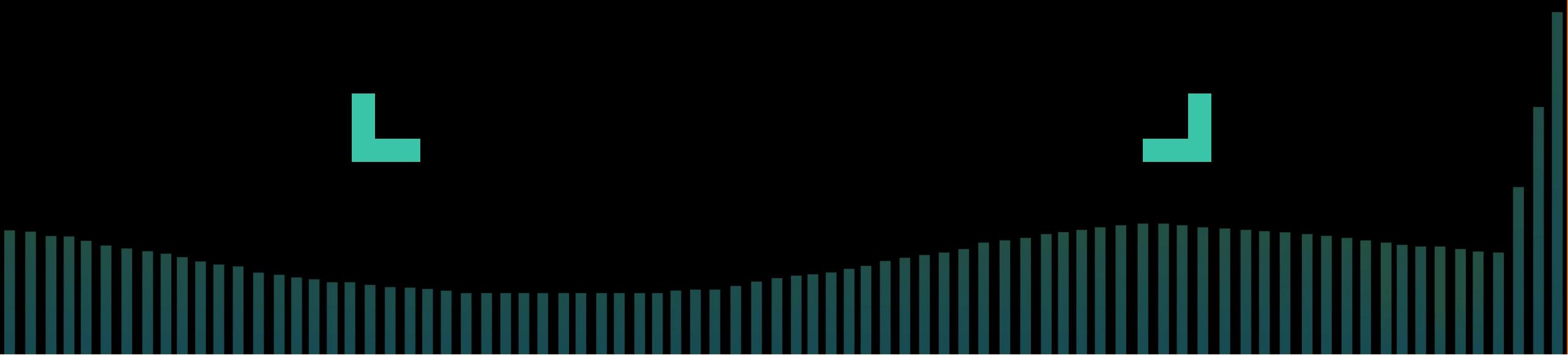
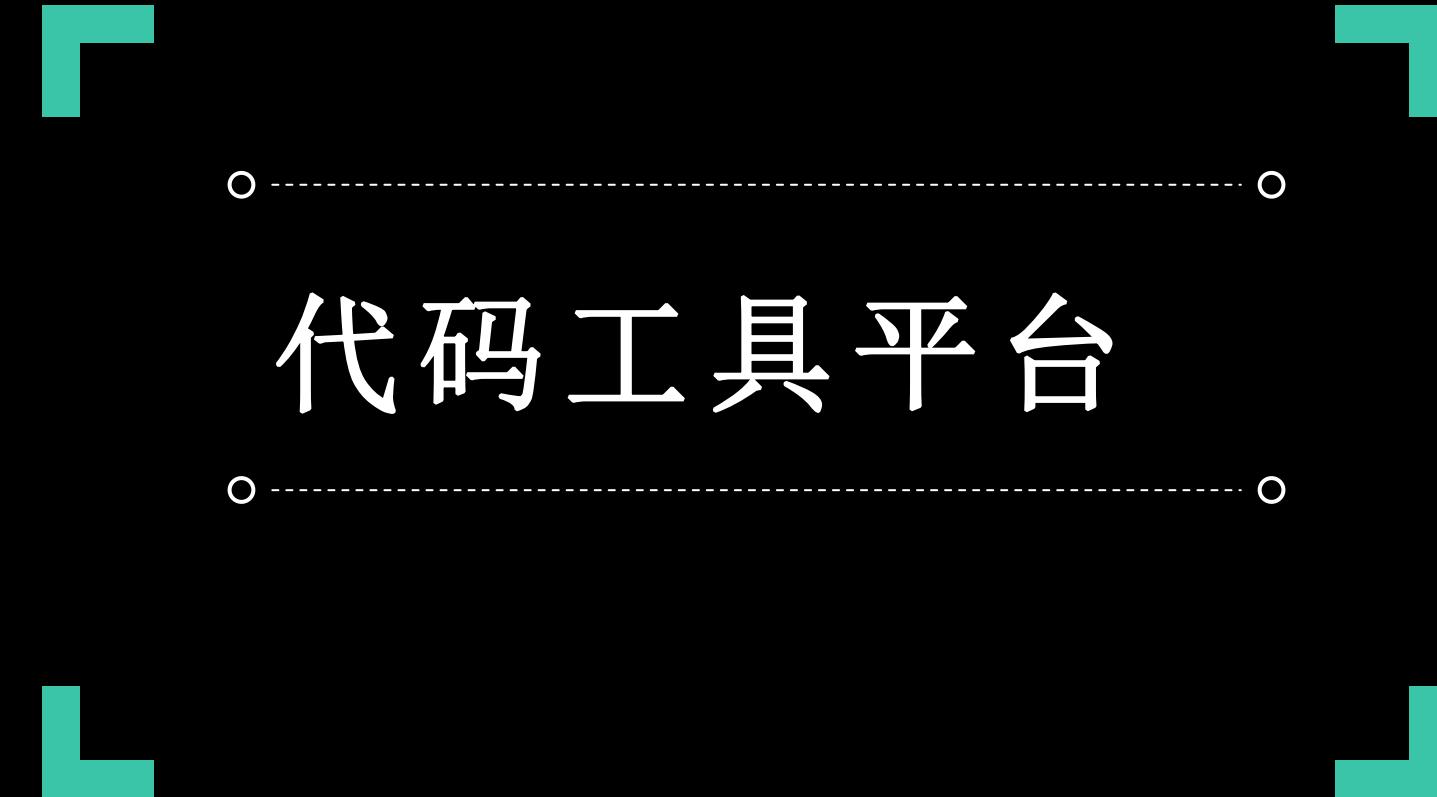
- string
- random/hash
- compress/crypto
- datetime
- hex/base64/json/xml/protobuf
- Threads/ThreadPool
- FileStorage
- Network
- HTTP/URL/QueryParams
- RPC

```
#ifndef COMMON_SYSTEM_NET_IP_ADDRESS_H_
#define COMMON_SYSTEM_NET_IP_ADDRESS_H_
#pragma once

#include <limits.h>
#include <stdio.h>
#include <string>
#include <vector>
#include "common/base/string/string_piece.h"
#include "common/system/net/os_socket.h"

namespace gdt {

    /// IP v4 address
    class Ipv4Address {
    public:
        static const size_t kByteSize = 4;
        constexpr Ipv4Address() : m_ip() {}
        constexpr explicit Ipv4Address(uint32_t ip);
        constexpr explicit Ipv4Address(in_addr addr);
        constexpr Ipv4Address(uint8_t b1, uint8_t b2,
                             uint8_t b3, uint8_t b4);
        explicit Ipv4Address(const char* src);
        explicit Ipv4Address(const std::string& src);
    };
}
```



代码平台工具——挑战和应对

- 挑战

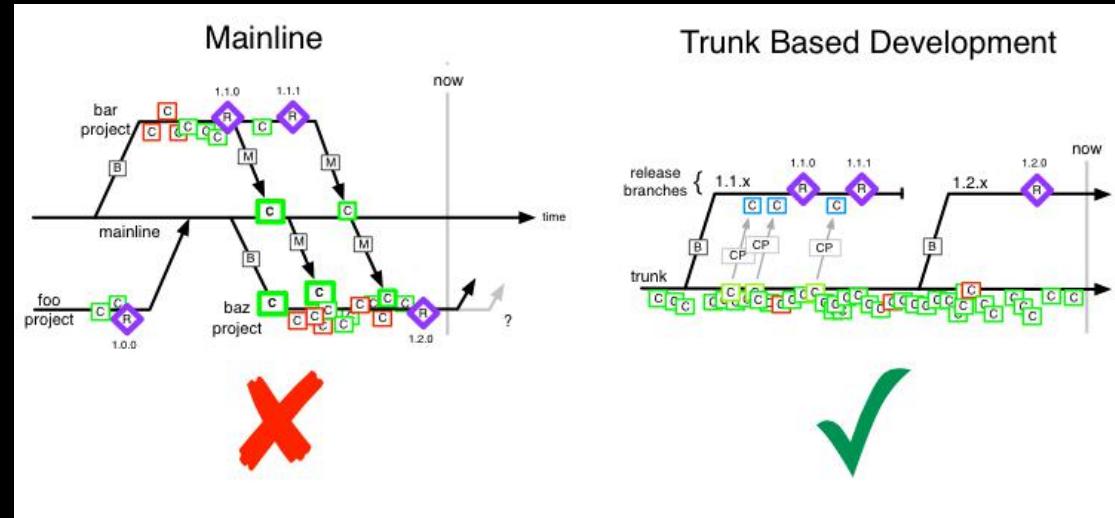
- 77377个C++文件，55911个Java文件
- 576万行C++代码
- 数百人在一个代码库上开发，直接发生相互影响
- master分支每天合并200多次，对CI压力很大
- 代码中存在着大量还在开发中的特性

- 应对

- 代码提交前增量检查
- 提交后的模块级代码质量检查
- 单元测试要求和覆盖率目标跟进
- 代码评审制度
- 特性开关系统
- 构建系统优化
- 持续集成系统
- 回归测试系统

代码平台工具——FeatureFlags控制特性变更

- 适配主干开发模式，减少合并冲突
- 方便实施自动化回归测试
- 方便对每个开关单独启用，灰度开启验证
- 方便有问题时单独关闭而不用回滚整个发布
- 可以精确统计和度量每个特性的访问情况
- 平台自动跟踪开关生命周期，并驱动清理



版本	flag	flag描述	下线时间
20201202 V1	fe_video_analysis	视频分析flag	2020-12-17
20201202 V1	switch_union_dev_api	切换联盟应用dev-api	2021-01-30
20201201 V2	fe_multiple_scene	通投版位多场景	2021-01-31
20201201 V1	fe_creative_table_new	创意工作台-新建视频	2021-11-01

代码工具平台——再谈构建系统

- 尽量从源代码构建
 - 减少二进制兼容问题
 - 方便编译器升级
- 不要用预编译头文件
 - 脆弱的实现
 - 增大不必要的耦合
- 静态链接发布和部署
 - 减少依赖
 - 方便编译器升级



A Public Service Announcement:

**Build Everything From
Source, All The Time**

A Case Study in Fear

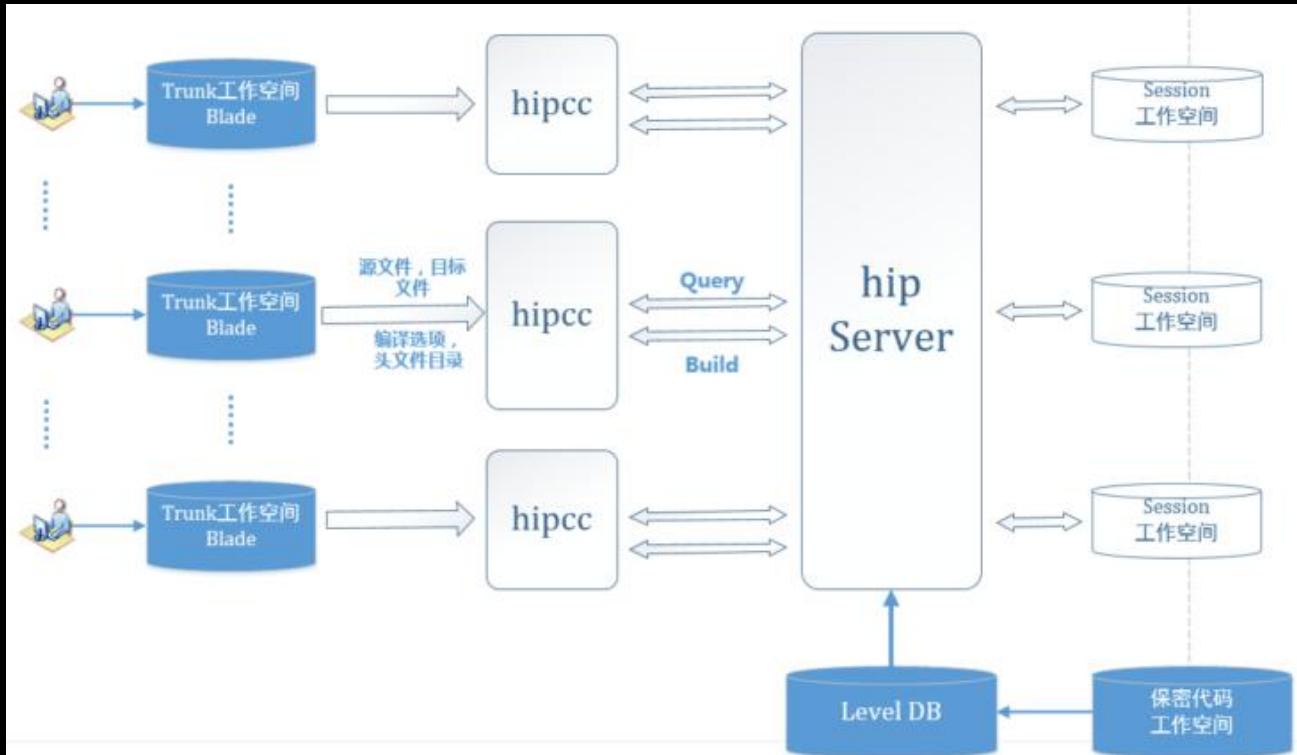
Dave Steffen, Ph.D.
Software Lead
dsteffen@scitec.com



代码工具平台——保密代码编译器

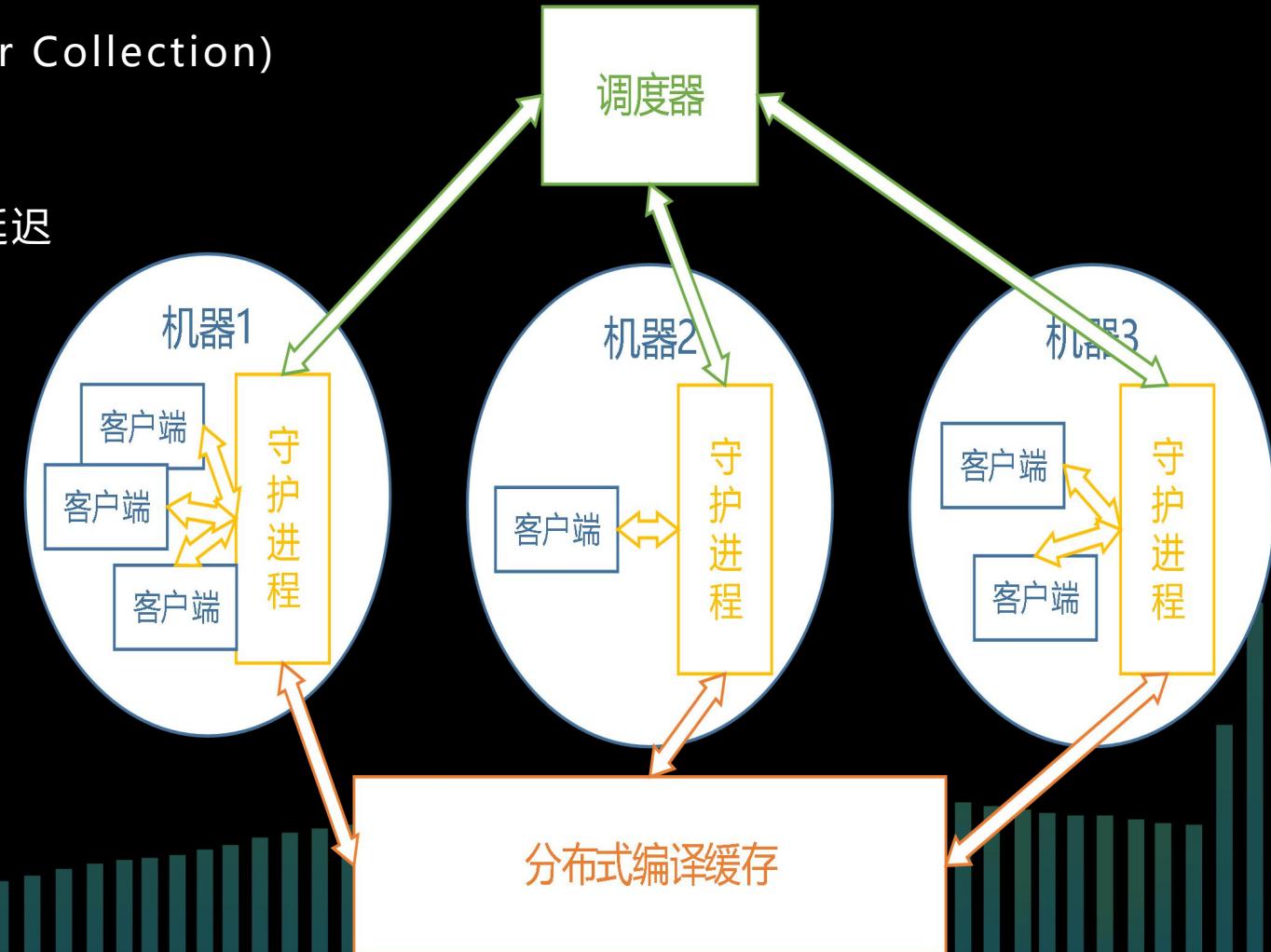
```
cc_library(  
    name = 'sunfish',  
    srcs = ['foo.cc', 'bar.cc', 'baz.cc'],  
    hip = True,  
    hip_version = 10777,  
)
```

- 99.9%的代码都是全员可以访问的
- 极少量代码，只有相关同事才能访问
- 建设保密代码构建系统构建这些代码



代码工具平台——分布式构建系统

- Yadcc (Yet Another Distributed Compiler Collection)
 - 中心化任务调度
 - 分布式缓存+布隆过滤器减少无效查询的延迟
 - 本地任务并发度控制
 - 用-fdirectives-only加速预处理
 - 使用高效率的压缩/哈希算法
 - “P2P” 方式共享CPU资源
 - 支持多版本编译器共存



代码工具平台——分布式构建系统

- 8C 2.5GHz Cascade Lake (KVM), 16 GB RAM, 集群1400核
- CentOS 7 / Devtoolset-9
- LLVM-project 11.0.0
- CXX=~/yadcc/symlinks/g++ CC=~/yadcc/symlinks/gcc cmake3 -G "Ninja" -DLLVM_ENABLE_PROJECTS="clang;libcxx;libcxxabi;libunwind;lldb;compiler-rt;lld" -DCMAKE_BUILD_TYPE=Release ..;/llvm
- time ninja -k256

```
[6124/6124] Creating executable symlink bin/clang
```

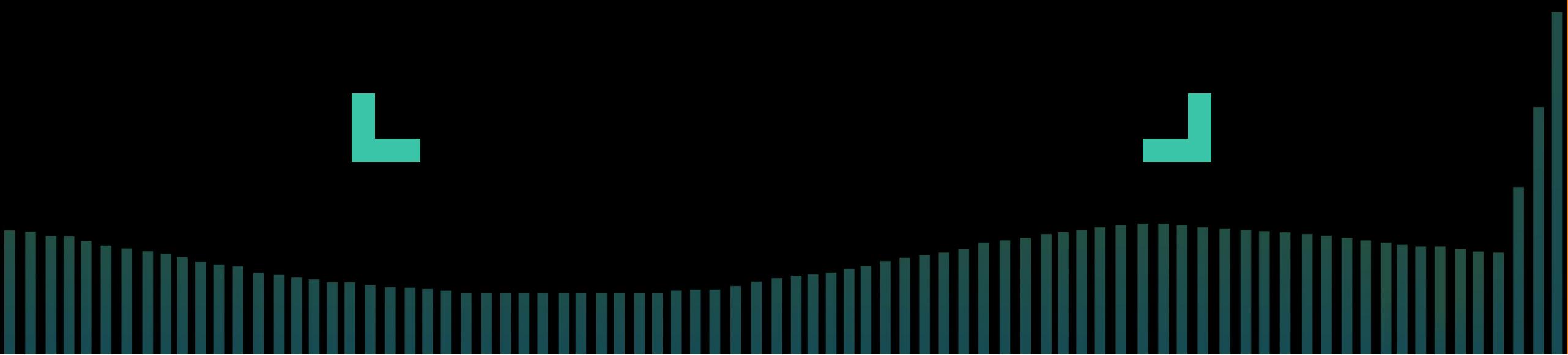
real	3m30.717s
user	18m44.844s
sys	4m50.628s

```
[6124/6124] Generating ../../bin/llvm-readelf
```

real	49m2.797s
user	366m6.104s
sys	22m3.883s

现代 C++

○ - ○
○ - ○



现代C++——编译器升级

- 单一代码库和静态链接使得升级更容易
- 升级过程中创建两个CI，确保都代码对两个编译器都兼容
- gnu++2a，全面支持C++17及部分C++20特性
- 提供代码检查工具引导使用新特性

年份	GCC版本	C++标准
2013	4.1.2	03
2015	4.4.6	0x
2016	4.9.4	14
2018	8.2	2a
2021?	11.2?	20

```
_test.cpp:16:tr1: Don't use std::tr1, use std library, see https://
_ test.cpp:17:tr1: Don't use std::tr1, use std library, see https://
_ test.cpp:20:tr1: Don't use std::tr1::unordered_map, use std::unor
```

```
_test.cpp:21:nullptr: Using nullptr instead of NULL, see https://
```

现代C++——Flare服务开发框架

- 基础库
- RPC框架



现代C++——Flare服务开发框架

- 基础库
 - 广泛使用标准库新增的类型改善可读性及性能
 - optional, chrono, string_view, move
 - 尚未被编译器支持的新标准库功能/好的提案：引入第三方库或仿造
 - fmt, expected
 - 补齐常用但标准库尚未支持的功能
 - base64, object pool, compression,...
 - 现代C++设计风格，通过Traits自定义/扩展库的行为

现代C++——Flare服务开发框架

```
// 旧式接口，用返回值报告错误，出参数返回实际结果
```

```
template <class T>
bool StringToNumeric(
    const std::string& s, T* result, int32_t base = 0);
```

```
template <class T, class = void>
struct TryParseTraits;
```

```
template <class T, class... Args>
inline std::optional<T> TryParse(const std::string_view& s,
                                  const Args&... args) {
    return TryParseTraits<T>::TryParse(s, args...);
}
```

```
template <class T>
struct TryParseTraits<T, std::enable_if_t<std::is_integral_v<T>>> {
    static std::optional<T> TryParse(
        const std::string_view& s, int base = 10);
};
```

现代C++——Flare服务开发框架

```
inline constexpr struct from_ip4_t {
    constexpr explicit from_ip4_t() = default;
} from_ip4;
inline constexpr struct from_ip6_t {
    constexpr explicit from_ip6_t() = default;
} from_ip6;

template <class T, class>
struct TryParseTraits;

template <>
struct TryParseTraits<Endpoint, void> {
    static std::optional<Endpoint> TryParse(const std::string_view& s,
                                              from_ip4_t);
    static std::optional<Endpoint> TryParse(const std::string_view& s,
                                              from_ip6_t);
    static std::optional<Endpoint> TryParse(const std::string_view& s);
};
```

现代C++——Flare服务开发框架

- RPC框架
 - 依赖注入支持多协议、调用链、监控、etc.
 - 底层基于用户态线程（Fiber）
 - 上层使用Future支持异步/异构/并发RPC
 - 利用现代C++改善接口
 - （后期）Future支持co_await：需要编译器支持

现代C++——Flare服务开发框架

```
EchoService_AsyncStub stub1("flare://some.polaris.address-1");
EchoService_AsyncStub stub2("flare://some.polaris.address-2");

EchoRequest req1, req2;
req1.set_body("body1");
req2.set_body("body2");

flare::RpcClientController ctr1, ctr2;
ctr1.SetTimeout(1s);
ctr2.SetTimeout(std::chrono::steady_clock::now() + 2s);
auto f1 = stub1.Echo(req1, &ctr1); // Future<Expected<EchoResponse, Status>>
auto f2 = stub2.Echo(req1, &ctr2);

auto&& [res1, res2] = flare::fiber::BlockingGet(flare::WhenAll(&f1, &f2));

if (res1 && res2) {
    FLARE_LOG_INFO("Received: {}, {}", res1->body(), res2->body());
}
```

总结

- 单一代码库和主干开发模式提高开发效率
- 通过工具化解决效率和质量之间的矛盾
- 跟进C++语言发展，走现代C++之路
- 面向现代C++建设基础库和框架，提升研发效率

Q

&

A



冯富秋

阿里云智能系统技术负责人



目前致力于阿里超大规模数据中心的稳定性和可靠性建设，新硬件与软件的协同设计等基础技术领域的研发与工程化落地。主要研究领域包括：SOC芯片的前端构建、仿真，电信级Linux内核研发，百万级系统智能运维，以及行业应用的全生栈软硬件技术及其融合应用。曾参与中国嵌入式系统学科体系建设，国家电网智能电网国家标准的制定。

主办方：

Boolan
高端IT咨询与教育平台

C++ Summit 2020

冯富秋

阿里云智能系统技术负责人

致力于阿里超大规模数据中心的稳定性和可靠性建设

Linux内核的语言虚拟机王者-EBPF

Today's agenda

- 1 Introduction
- 2 How to do tracing in ultra-large-scale data centers
- 3 cBPF Introduction, History & Implementation
- 4 eBPF Introduction, History & Implementation
- 5 eBPF Tracing and Monitoring
- 6 Alibaba Tracing and Monitoring SIG

1

Introduction

\$ WHOAMI
冯富秋 (Tinnal)



- Senior Technical Expert, Team Leader @ Alibaba
- System Technology Team
- What I do:
 - Build customer competitiveness by OS
 - Stability and reliability construction of ultra-large-scale data centers.

2

How to do Tracing/Monitoring in ultra-large-scale data centers

2.1 How to do tracing in ultra-large-scale data centers

Runtime environment of ultra-large-scale data centers

- Heterogeneous computer architecture
- Ultra high reliability requirements
- Must have minimal overhead
- Static mix dynamic tracing/monitoring instrumentation

2.2 Traditional dynamic Tracing

Tracer/Front-end	Data sources	Data collection
ftrace	kprobe, uprobe, tracepoints, USDT	ftrace (sysfs)
Perf	perf_events (+kprobes, tracepoints...)	perf ring buffer
SystemTap	kprobe, uprobe, tracepoints, USDT Specific events, or user space tracing syscalls (not sure how)	kernel module
LTTng	DTrace probes... but not on Linux!	kernel module
Sysdig		Sysdig ring buffer
DTrace		

Limitations:

- SystemTap and LTTng require building and inserting kernel modules
- ftrace, perf, LTTng lack programmability
- SystemTap not user friendly, could crash the kernel
- Sysdig limited to system calls
- DTrace very powerful but not in Linux kernel (Some out-of-tree ports available)

3

cBPF Introduction, History & Implementation

3.1 cBPF Introduction

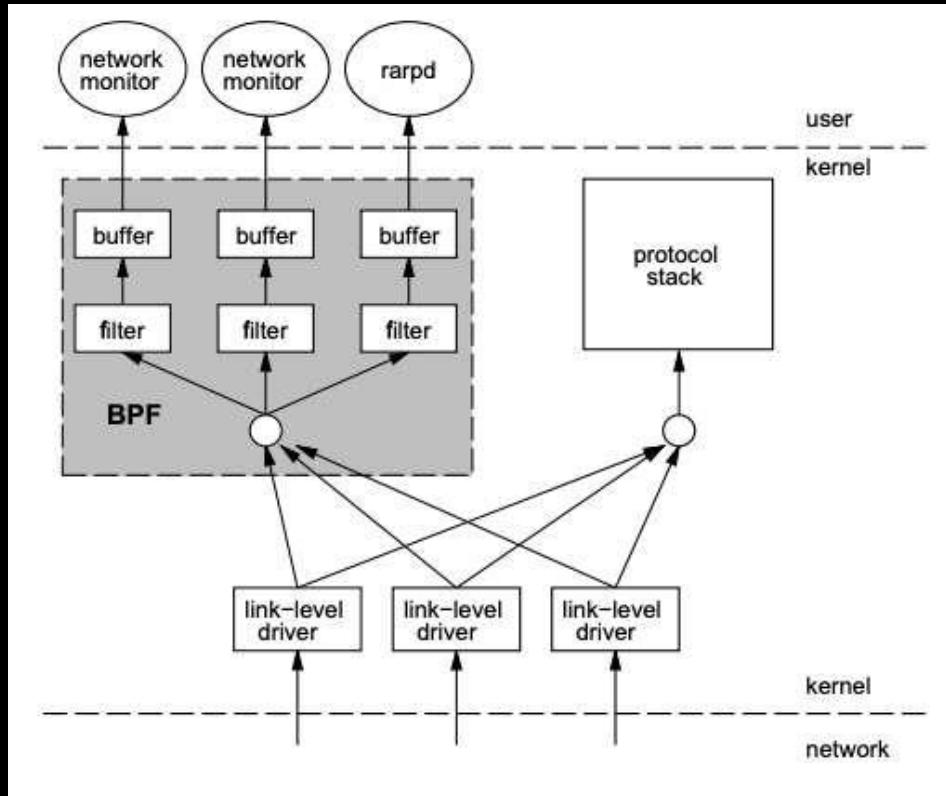
What is cBPF?

- cBPF – Classic BPF
 - Also known as “Linux Packet Filtering”
- Network packet filtering, Seccomp
- Filter Expression → Bytecode → Interpret
- Small, in-kernel VM, Register based, few instructions

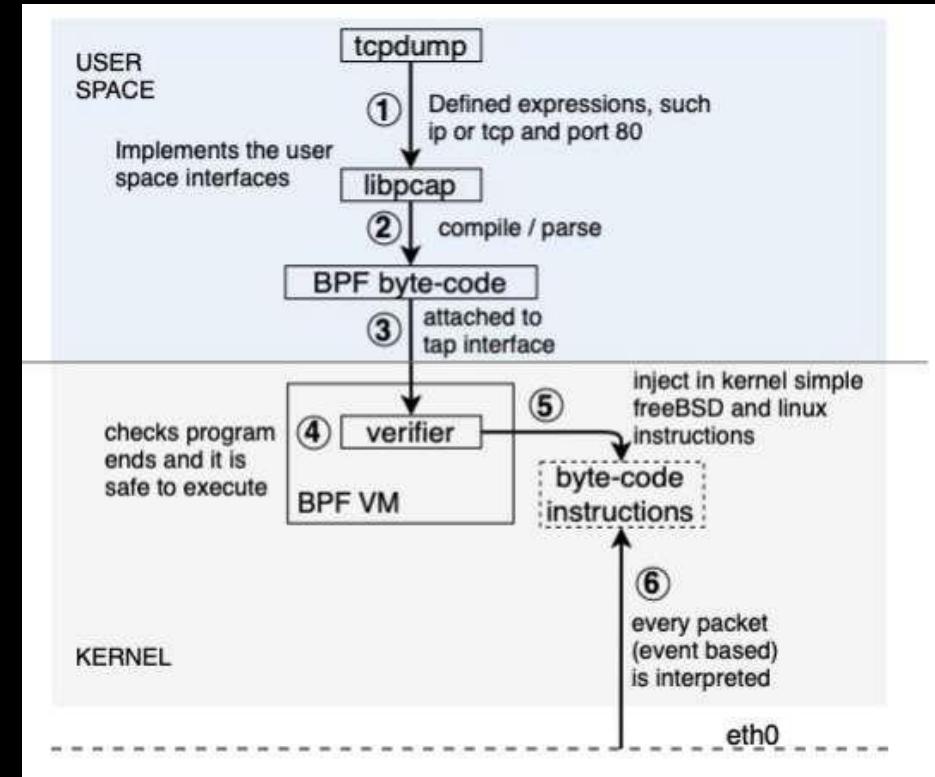
3.2 History of BPF

- Before BPF, each OS (Sun, DEC, SGI etc) had its own packet filtering API
- In 1993: Steven McCanne & Van Jacobson released a paper titled the *BSD Packet Filter (BPF)*
- In 1997: Implemented as “Linux Socket Filter” in kernel 2.1.75
- While maintaining the BPF language (for describing filters), uses a different internal architecture

3.3 BPF implementation



<https://www.tcpdump.org/papers/bpf-usenix93.pdf>



https://static.sched.com/hosted_files/kccnceu19/b8/KubeCon-Europe-2019-Beatriz_Martinez_eBPF.pdf

3.3 BPF implementation

Structure

```
struct sock_filter { /* Filter block */
    __u16 code; /* Actual filter code */
    __u8 jt; /* Jump true */
    __u8jf; /* Jump false */
    __u32 k; /* Generic multiuse field */};
```

characteristic

- two 32-bit registers: A, X
- 16 * 32-bit slots stack
- full integer arithmetic
- explicit load/store from packet
- conditional branches (with two destinations: jump true/false)

3.3 BPF Implementation

Instruction

Inst.	AddrMode	Description
ld	1, 2, 3, 4, 12	Load word into A
<...>		
Idx	3, 4, 5, 12	Load word into X
<...>		
st	3	Store A into M[]
stx	3	Store X into M[]
jmp	6	Jump to label
ja	6	Jump to label
jeq	7, 8, 9, 10	Jump on A == <x>
<...>		
add	0, 4	A + <x>
sub	0, 4	A - <x>
<...>		
tax		Copy A into X
txa		Copy X into A
ret	4, 11	Return

Addressing mode

AddrMode	Syntax	Description
0	x/%x	Register X
1	[k]	BHW at byte offset k in the packet
2	[x + k]	BHW at the offset X + k in the packet
3	M[k]	Word at offset k in M[]
4	#k	Literal value stored in k
5	4*([k]&0xf)	Lower nibble * 4 at byte offset k in the packet
6	L	Jump label L
7	#k,Lt,Lf	Jump to Lt if true, otherwise jump to Lf
8	x/%x,Lt,Lf	Jump to Lt if true, otherwise jump to Lf
9	#k,Lt	Jump to Lt if predicate is true
10	x/%x,Lt	Jump to Lt if predicate is true
11	a/%a	Accumulator A
12	extension	BPF extension

3.3 BPF Implementation

```
# tcpdump -i eth0 tcp dst port 22 -d
```

```
(000) ldh [12]          # Ethertype
(001) jeq #0x86dd jt 2 jf 6   # is IPv6?
(002) ldb [20]          # IPv6 next header field
(003) jeq #0x6 jt 4 jf 15    # is TCP?
(004) ldh [56]          # TCP dst port
(005) jeq #0x16 jt 14 jf 15   # is port 22?
(006) jeq #0x800 jt 7 jf 15    # is IPv4?
(007) ldb [23]          # IPv4 protocol field
(008) jeq #0x6 jt 9 jf 15    # is TCP?
(009) ldh [20]          # IPv4 flags + frag. offset
(010) jset #0xffff jt 15 jf 11   # fragment offset is != 0?
(011) ldxb 4*([14]&0xf)      # x := 4 * header_length (words)
(012) ldh [x + 16]        # TCP dst port
(013) jeq #0x16 jt 14 jf 15    # is port 22?
(014) ret #262144         # trim to 262144 bytes, return packet
(015) ret #0              # drop packet
```

4

eBPF Introduction, History & Implementation

4.1 eBPF Introduction



“crazy stuff”

- **Alexei Starovoitov (eBPF lead)**



“eBPF is Linux’s new superpower”

- **Gaurav Gupta**



“eBPF does to Linux what JavaScript does to HTML”

— **Brendan Gregg**

4.1 eBPF Introduction

What is eBPF?

- eBPF – extended Berkeley Packet Filter
- User-defined, sandboxed bytecode executed by the kernel
- VM that implements a RISC-like assembly language in kernel space
- All interactions between kernel/ user space are done through eBPF “maps”
- eBPF does not allow loops

4.1 eBPF Introduction

What can BPF be used for

NETWORKING

- Load-balancing
 - Katran(Facebook)
- General networking
 - Cilium
- Extending the TCP stack
- Network Monitoring
 - Flowmill
 - Weaveworks

FIREWALLS

BpfILTER (Linux 4.18)

PROFILE >< TRACING

- Sysdig
- bpftrace

DDOS MITIGATION

- Use of eBPF & XDP to perform infra-wide DDoS mitigation
- Facebook
 - Cloudflare

CHAOS ENGINEERING

Use Cilium to inject latency, packet-loss, L7 HTTP errors (via a Go extension)

DEVICE DRIVERS

eBPF provides a pseudo device driver possible to extend this in multiple ways

4.2 History of eBPF

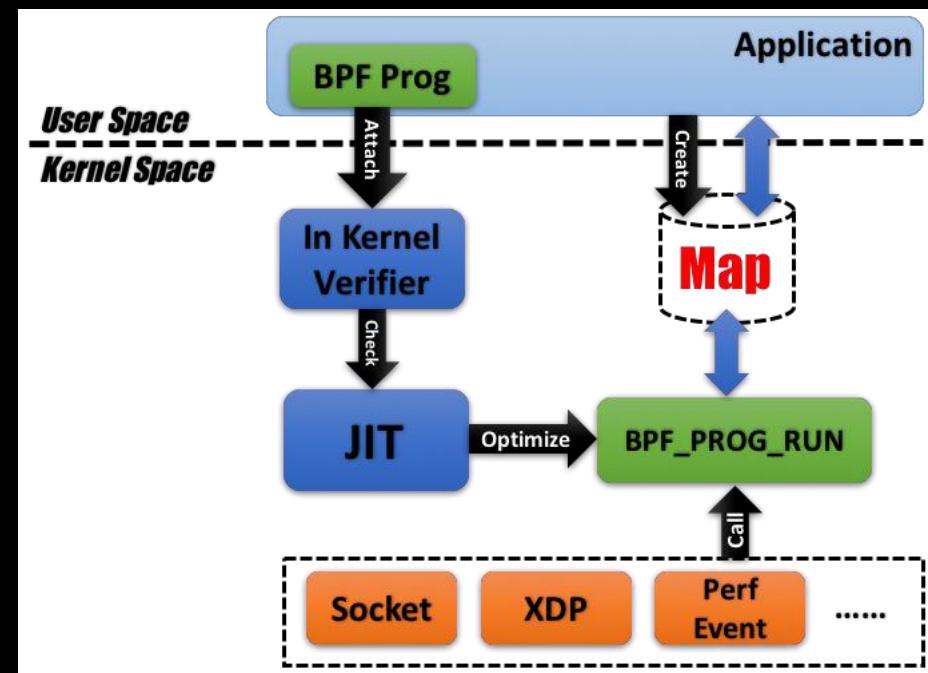
- Development started in 2011 - A JIT for packet filters
- Add tracing filters with BPF in 2013
- eBPF first included with kernel 3.18 in 2014
- BPF backend for LLVM upstreamed in Jan 2015

4.3 eBPF implementation

design goals

- parse, lookup, update, modify network packets
- loadable as kernel modules on demand, on live traffic
- safe on production system
- performance equal to native x86 code
- fast interpreter speed (good performance on all architectures)
- calls into bpf and calls from bpf to kernel should be free (no FFI overhead)

Architecture of eBPF



4.3 eBPF implementation

Design Intent

- take a mix of real CPU instructions
(10% classic BPF + 70% x86 + 25% arm64 + 5% risc)
- analyze x86/arm64/risc calling conventions and define a common one for this 'renamed' instruction set
- make instruction encoding fixed size (for high interpreter speed)
- reuse classic BPF instruction encoding (for trivial classic->extended conversion)

4.3 eBPF implementation

Core changes of the new internal format

- Number of registers increase from 2 to 10 (+1 stack register)
- Register width increases from 32-bit to 64-bit
- Conditional jt/jf targets replaced with jt/fall-through
- Introduces bpf_call insn and register passing convention for zero overhead calls from/to other kernel functions

4.3 eBPF implementation

Performance

Just-In-Time compilation: BPF instructions → native code

- Alternative to kernel interpreter, brings performance
- Supported architectures:
ARM32, ARM64, MIPS, PowerPC64, RiscV, Sparc64, s390, x86_32,
x86_64
- Hardware offload: NFP (Netronome)
- all registers map one-to-one
- most of instructions map one-to-one
- bpf ‘call’ instruction maps to x86 ‘call’

4.3 eBPF Implementation

Verifier – Ensure Termination and Safety

BPF programs come from user space: make sure they terminate / are safe

Termination:

- Direct Acyclic Graph (DAG), inspect instructions, prune safe paths Maximum: 4096 instructions... OH WAIT now “up to 1 million” for root No back edge (loops)
- Except function calls
- May change soon (bounded loops)

Safety:

- No unreachable code, no jump out of range
- Registers and stack states are valid for every instruction
- Program does not read uninitialised registers/memory
- Program only interacts with relevant context (prevent out-of-bound/random memory access)

4.3 eBPF Implementation

Verifier – Ensure Termination and Safety

"One of the more interesting features in this cycle is the ability to attach eBPF programs (**user-defined, sandboxed bytecode executed by the kernel**) to kprobes. This allows user- defined instrumentation on a live kernel image that can never crash, hang or interfere with the kernel negatively."

– Ingo Molnár (Linux developer)

Source: <https://lkml.org/lkml/2015/4/14/232>

4.3 eBPF implementation

BPF calling convention

BPF calling convention was carefully selected to match a subset of amd64/arm64 ABIs to avoid extra copy in calls:

- R0 – return value
- R1..R5 – function arguments
- R6..R9 – callee saved
- R10 – frame pointer

X86 Mapping

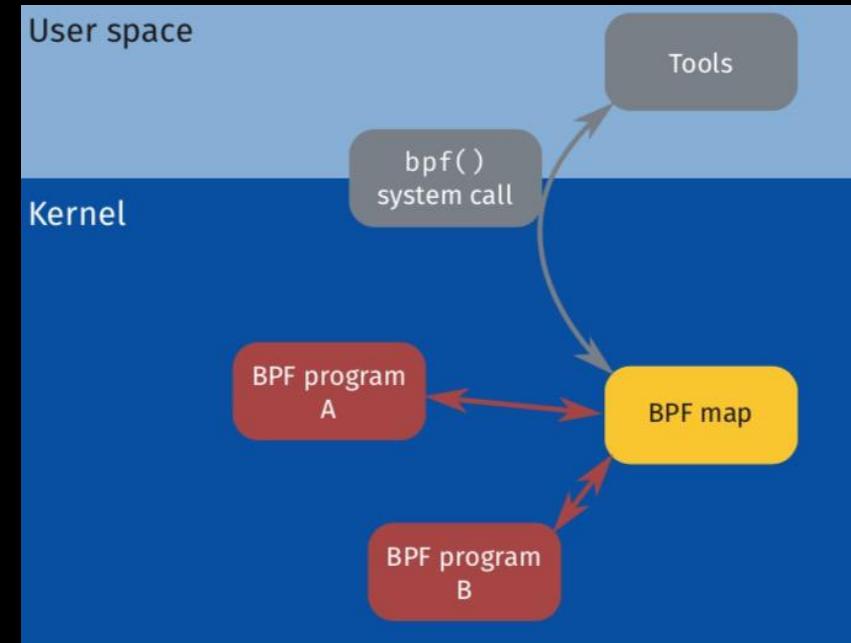
R0	-	rax
R1	-	rdi
R2	-	rsi
R3	-	rdx
R4	-	rcx
R5	-	r8
R6	-	rbx
R7	-	r13
R8	-	r14
R9	-	r15
R10	-	rbp

4.3 eBPF implementation

BPF maps

“Maps”: special kernel memory area accessible to a program

- Shared between: several program runs, several programs, user space
- Typically, “key/value” storage: hash map, array
- Some of them have a “per-CPU” version
- Generally, RCU-protected; also, spinlocks now available in BPF



4.3 eBPF implementation

BPF maps

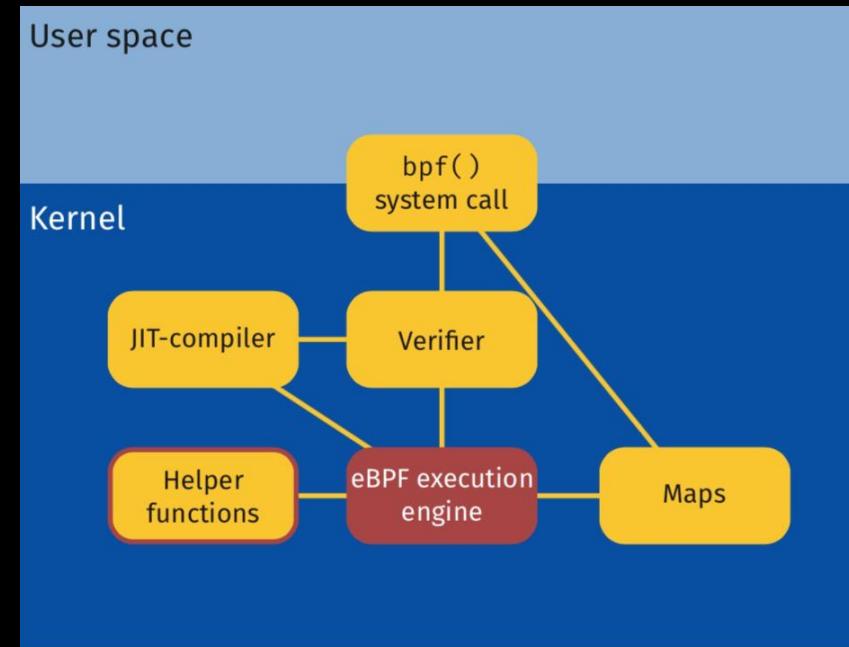
“Standard library” of functions implemented in the kernel

- Can be called from BPF programs
Ease some tasks, manipulate maps, context, ...,
e.g.:

- Map lookup, update
- Get kernel time
- printk() equivalent
- Change packet size
- Redirect a packet
- Safely dereference a kernel pointer

- Up to 5 arguments
Some of them restricted to GPL-compatible BPF
programs

More than 100 helper functions in the kernel
already



4.3 eBPF implementation

Extended BPF assembler

```
int bpf_prog(struct bpf_context *ctx)
{
    u64 loc = ctx->arg2;
    u64 init_val = 1;
    u64 *value;

    value = bpf_map_lookup_elem(&my_map, &loc);
    if (value)
        *value += 1;
    else
        bpf_map_update_elem(&my_map, &loc,
                            &init_val, BPF_ANY);
    return 0;
}
```

compiled by LLVM from C to bpf asm

```
1: *(u64 *)(r10 -8) = r1
2: r1 = 1
3: *(u64 *)(r10 -16) = r1
4: r1 = map_fd
5: r2 = r10
6: r2 += -8
7: call 1
8: if r0 == 0x0 goto pc +4
9: r1 = *(u64 *)(r0 +0)
10: r1 += 1
11: *(u64 *)(r0 + 0) = r1
12: goto pc+8
13: r1 = map_fd
14: r2 = r10
15: r2 += -8
16: r3 = r10
17: r3 += -16
18: r4 = 0
19: call 2
20: r0 = 0
21: exit
```

4.3 eBPF implementation

BPF compilers

- BPF backend for LLVM is in trunk and will be released as part of 3.7
- BPF backend for GCC is being worked on
- C front-end (clang) is used today to compile C code into BPF
- tracing and networking use cases may need custom languages
- BPF backend only knows how to emit instructions (calls to helper functions look like normal calls)

```
clang -O2 -g -emit-llvm -c prog.c -o - | \ llc -  
march=bpf -mcpu=probe -filetype=obj -o prog.o
```

Other alternatives: Lua, Rust, ...

5

ebpf Tracing and Monitoring

5.1 BPF for Tracing

BPF can attach to:

- kprobes/kretprobes
- uprobes/uretprobes
- tracepoints, USDT
- perf_events

Data collection:

- ftrace/sysfs
- perf ring buffer
- BPF maps

Advantages:

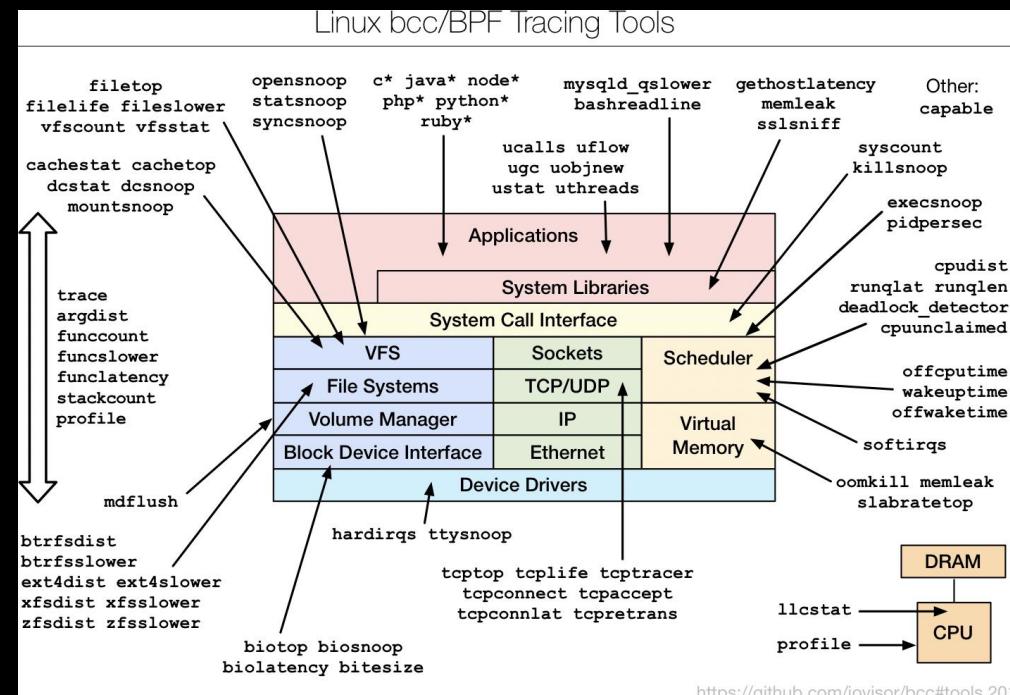
- Programmable
- Fast, secure
- No kernel module

5.2 BPF Tracing Tools: bcc

bcc: Framework for BPF tools, mostly a set of Python wrappers

- Framework for BPF tools, mostly a set of Python wrapper.
- Also comes with a set of tools (87 tracing tools for now, +examples)

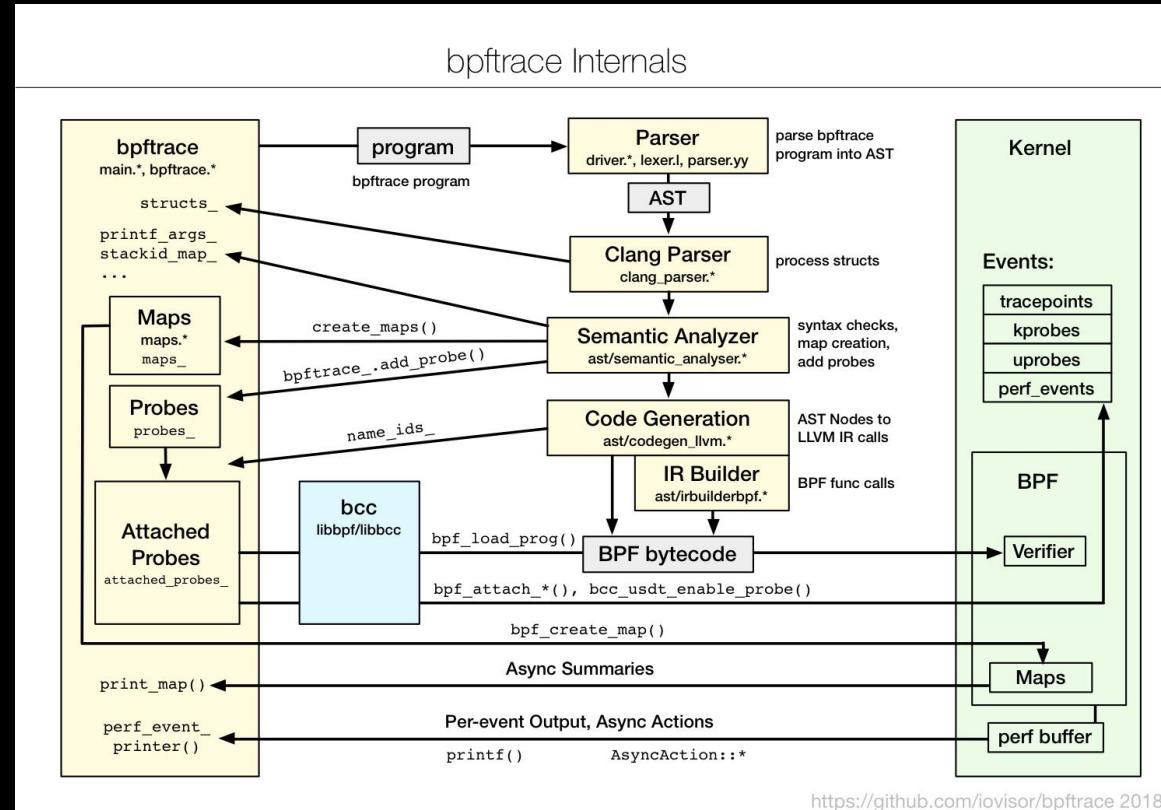
```
from bcc import BPF
b = BPF(text="""
#include <uapi/linux/ptrace.h>
int trace_open(struct pt_regs *ctx, int dfd,
               const char __user *filename, int flags)
{
    u64 id = bpf_get_current_pid_tgid();
    u32 pid = id >> 32;
    bpf_trace_printk("%d: open(%s, %x)\\n",
                     pid, filename, flags);
    return 0;
}
""")
b.attach_kprobe(event="do_sys_open",
                fn_name="trace_open").trace_print()
```



5.3 BPF Tracing Tools: bpftrace

bpftrace is a high-level tracing language for Linux enhanced Berkeley Packet Filter (eBPF) available in recent Linux kernels (4.x)

- Awk-like syntax
- Sits on top of bcc
- Embeds a number of built-in functions and variables “Linux equivalent to DTrace”
- Express programs as one-liners, or very short scripts
- Also comes with a number of ready-to-use scripts
- (Has very good documentation!)



Our simple example to monitor open():

```
↳ bpftrace -e 'kprobe:do_sys_open { printf("%d-%s: %s\n", pid, comm, str(arg1)) }'
```

6

Alibaba Tracing and Monitoring SIG

阿里巴巴跟踪诊断技术SIG(特别兴趣组)

小组简介

跟踪诊断技术是操作系统中必不可少的基础能力。在Alibaba Cloud Linux系统开源实践过程中，既有基于 eBPF 技术的bcc 工具集、NX 套件，又有内核各个子系统的 SLI 和 tracing 框架，也有在实际业务生产场景中落地的 Diagnose-tool。这些工具和能力都将在跟踪诊断技术 SIG 中呈现。

小组目标

为Alibaba Cloud Linux开源操作系统，提供一个全栈覆盖内核与核心组件的跟踪和诊断工具、平台，增强 OpenAnolis 全栈的可观察性与可靠性。



6.1 Alibaba Tracing and Monitoring SIG

胡俊锋（岗德）

阿里云智能IoT操作系统AliOS Things总负责人



目前负责阿里云IoT智能设备平台研发部，是IoT端上操作系统AliOS Things的总负责人，提出HaaS并负责0到1落地。在此之前于阿里巴巴人工智能实验室负责天猫精灵系统软件和AIoT总体架构设计，是蓝牙mesh从0到1落地天猫精灵开放平台的负责人。十多年嵌入式操作系统设计和研发经验，熟悉RTOS、Linux、微内核。是蓝牙BLE&mesh方面的资深专家，出版第一本中文蓝牙mesh书籍《蓝牙mesh实战》。拥有累计30多项的物联网专利。

主办方：

Boolan
高端IT咨询与教育平台

AliOS Things物联网操作系统 架构设计

固德

阿里云智能 资深技术专家

2020.12

嵌入式操作系统的发展历史



嵌入式操作系统的三大痛点



碎片化



低安全



弱交互

AliOS Things物联网操作系统

背景

- 始于2016年
- 阿里云IoT核心团队自主研发
- 已大量应用

定位

- 以实时操作系统为基础
- 弹性支持通用OS能力组件
- 主打连接上云、多模态交互、AIoT小程序生态

价值

- 国产自主可控
- 降低物联网设备成本
- 减少研发时间成本
- 促进物联网生态发展

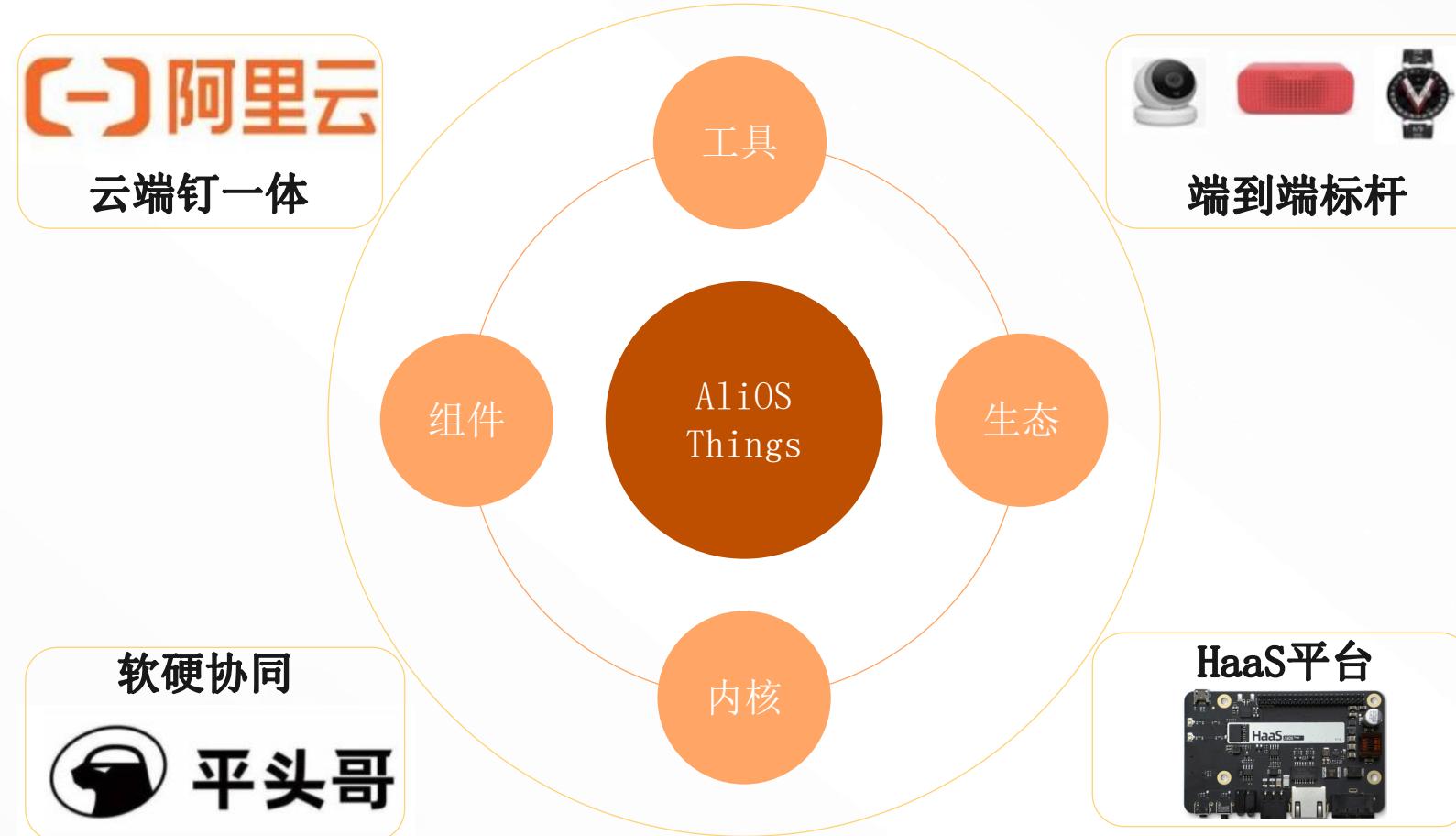
AliOS Things物联网操作系统版本演进



AliOS Things物联网操作系统典型应用



AliOS Things物联网操作系统生态完善



生态完善

- 一环
 - ✓ 内核：领先的内核技术
 - ✓ 组件：组件丰富，兼容POSIX接口
 - ✓ 工具：统一IDE、SmartTrace、JS拖拽低代码开发、Web远程运维等
 - ✓ 生态：开源github、捐献给开放原子开源基金会共建，开发者门户
- 二环
 - ✓ 软硬协同：IP Core + AliOS Things + 工具一体化输出
 - ✓ 云端一体：自动连云，钉钉数据运营入口
 - ✓ 标杆：打造端到端标杆
 - ✓ HaaS平台：业内首推HaaS平台，软硬件积木化，自研和认证硬件模组、开发板等

AliOS Things物联网操作系统服务聚合



阿里巴巴经济体服务

 阿里云

AliOS Things

软硬
协同

平头哥
芯片商
板卡商

开发者生态

 Cainiao 菜鸟 钉钉 天猫精灵
TMALL GENIE

云端钉
一体

AliOS Things物联网操作系统行业影响

AliOS Things服务大类设备140+, 应用组件300+



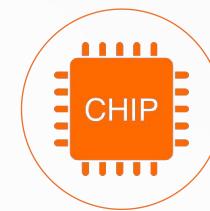
GitHub Star
3.6K



GitHub Fork
1.4K



协议栈数量
57



适配芯片数
芯片400+, 传感器100+



开发者数
30万+

接口统一

- POSIX标准接口
- 通用硬件抽象设计

组件丰富

- 全连接协议
- OTA
- AI框架
- 安全框架
- 传感器框架
- 语音框架

开发便利

- 自主开源
- 工具丰富
- 文档齐全

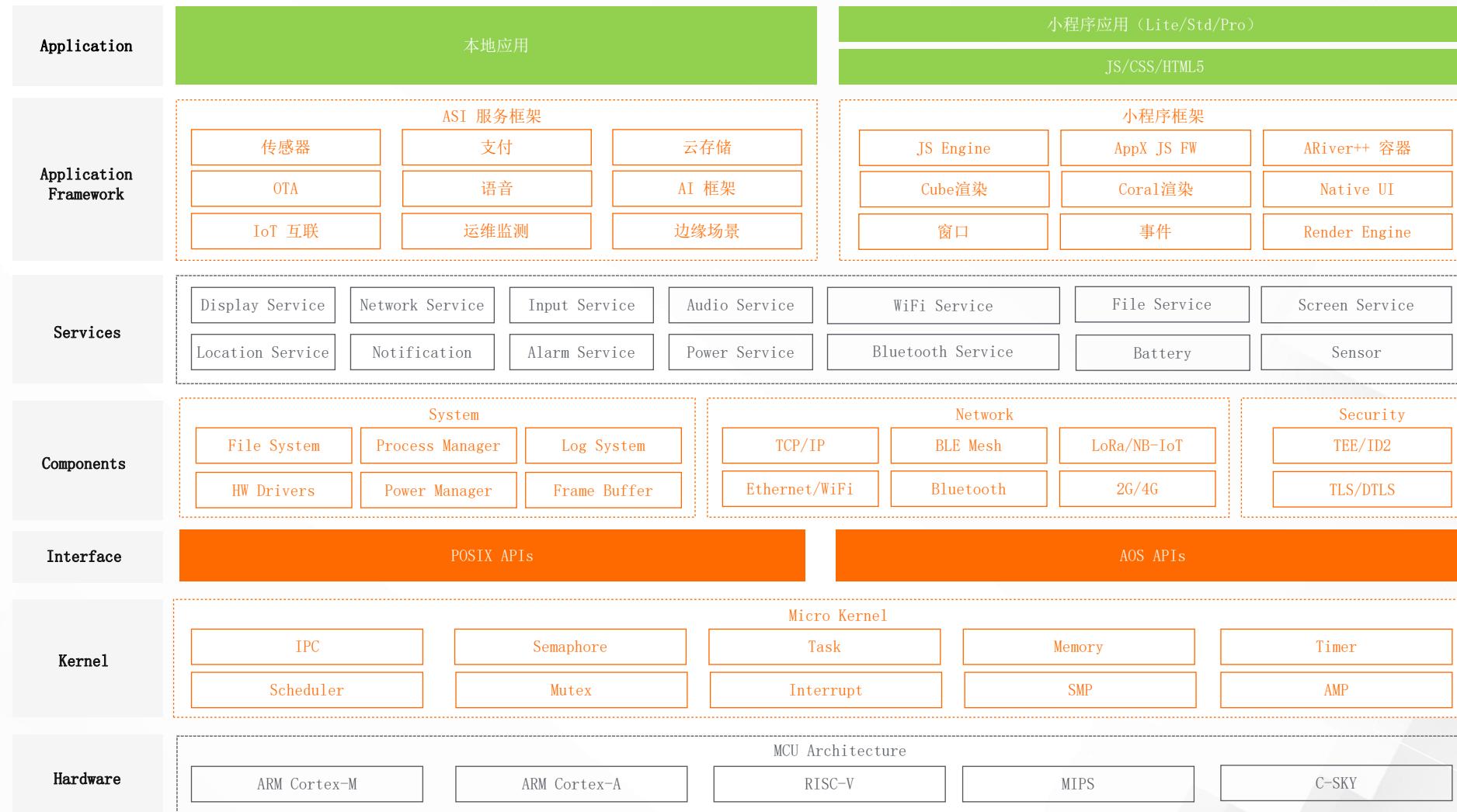
AliOS Things物联网操作系统捐赠共建

[主页](#)[项目](#)[合作者关系](#)[社区生态](#)[关于我们](#)[注册](#)

AliOS Things

[下载项目](#)[项目代码](#)[项目社区](#)[项目部署](#)

AliOS Things物联网操作系统架构设计



四大核心优势



微内核

- 系统可伸缩性强解决碎片化
- 内核对象细粒度安全可控
-



全面支持小程序

- 一次开发，多端投放
- 兼容支付宝小程序生态
-



自主知识产权

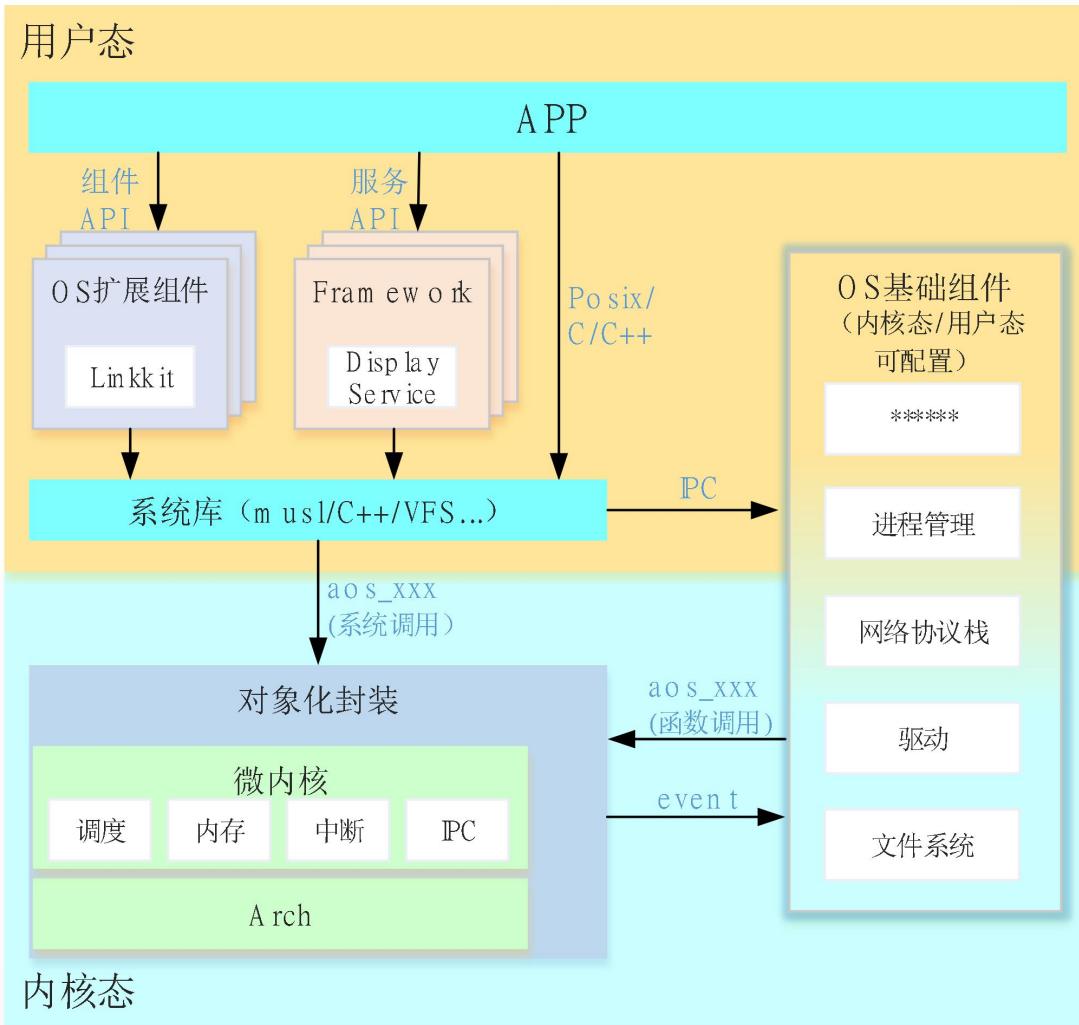
- Apache 2.0
- 无License污染问题
-



端云一体开发模式

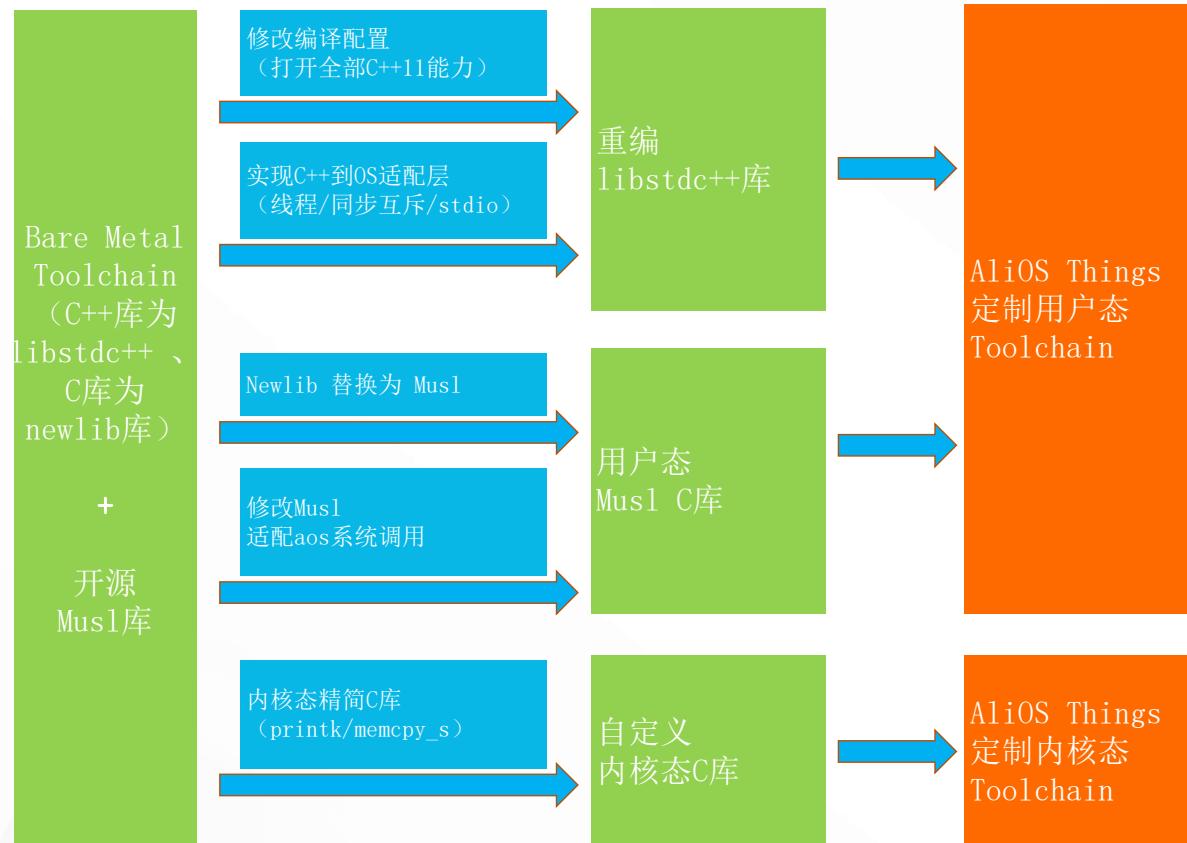
- 统一友好的IDE开发环境，极简代码开发
- 丰富的调试诊断工具
-

内核弹性可伸缩设计



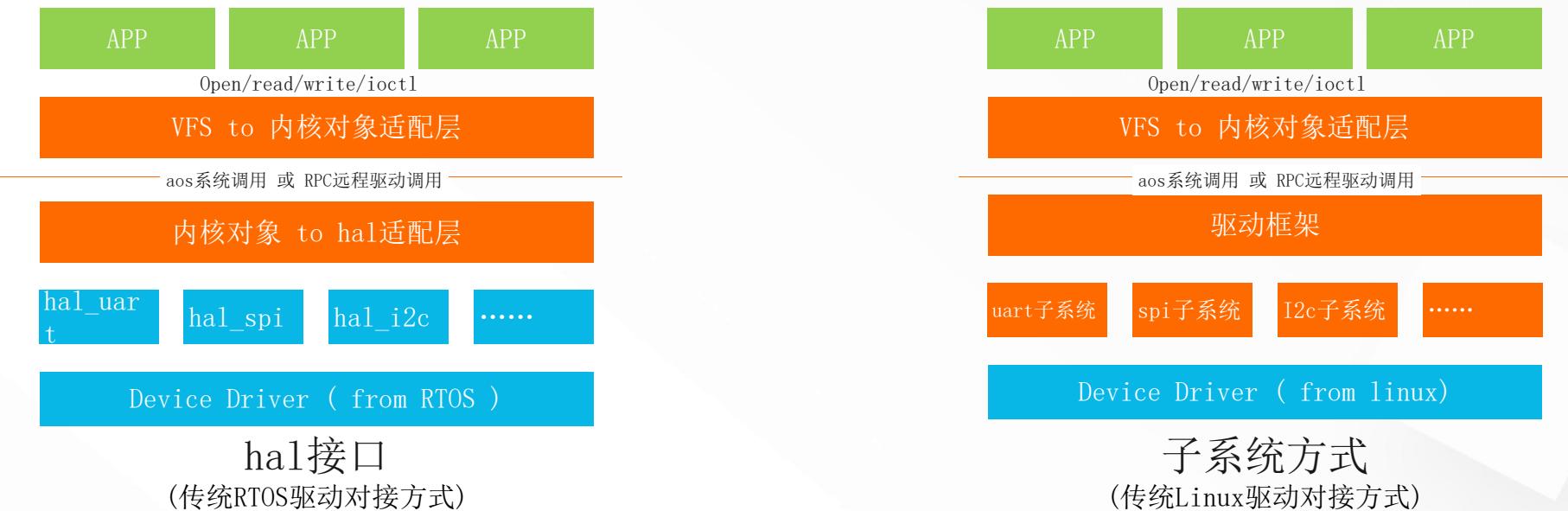
- AliOS Things能力可上（用户态）可下（内核态）：
 - 轻量化微内核，进行对象化语义封装出“aos_XXX” API；
 - 系统调用延续“aos_XXX” API，内核对外界面统一；
 - 高效的IPC机制；
 - OS基础组件能力可上可下，灵活部署；
- 系统组件和服务高度模块化，功能可以灵活配置；
- 内核高度精简，footprint在100KB之内，可以支持从低端到高端各种芯片；
- 应用、组件、驱动之间低耦合、高容错，易于开发维护；

应用兼容式设计



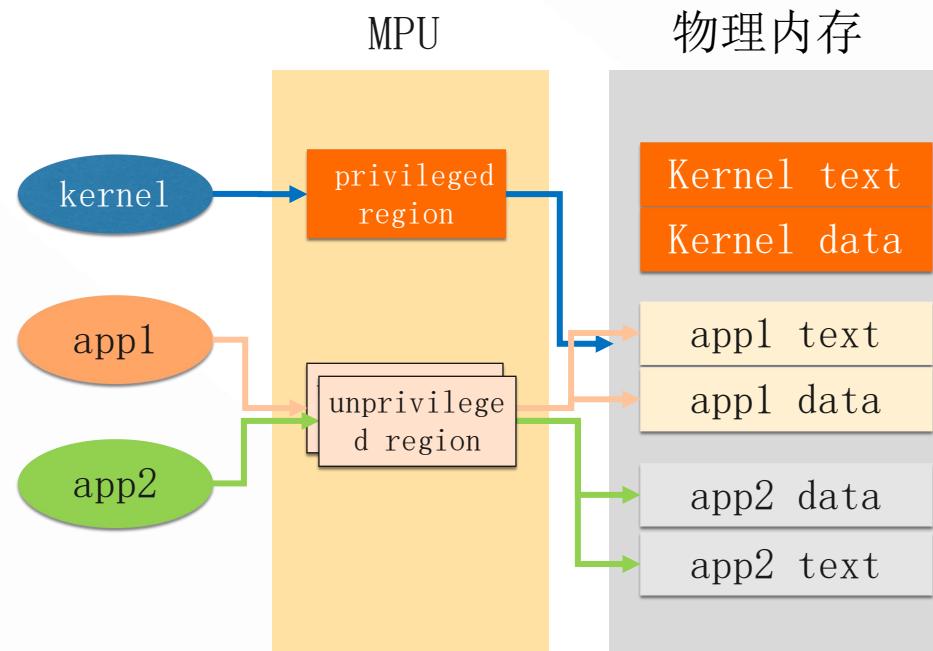
- 基于ARM bare metal工具链，借力开源Musl C库，定制出AliOS Things配套工具链
 - 支持700+的POSIX接口，方便用户态程序移植
 - 支持全量C++11特性，如thread线程类、原子操作、条件变量、智能指针、线程本地变量等
 - 支持多线程友好的C库
 - 增加各种safe版本C库接口，提高安全性

驱动兼容式设计

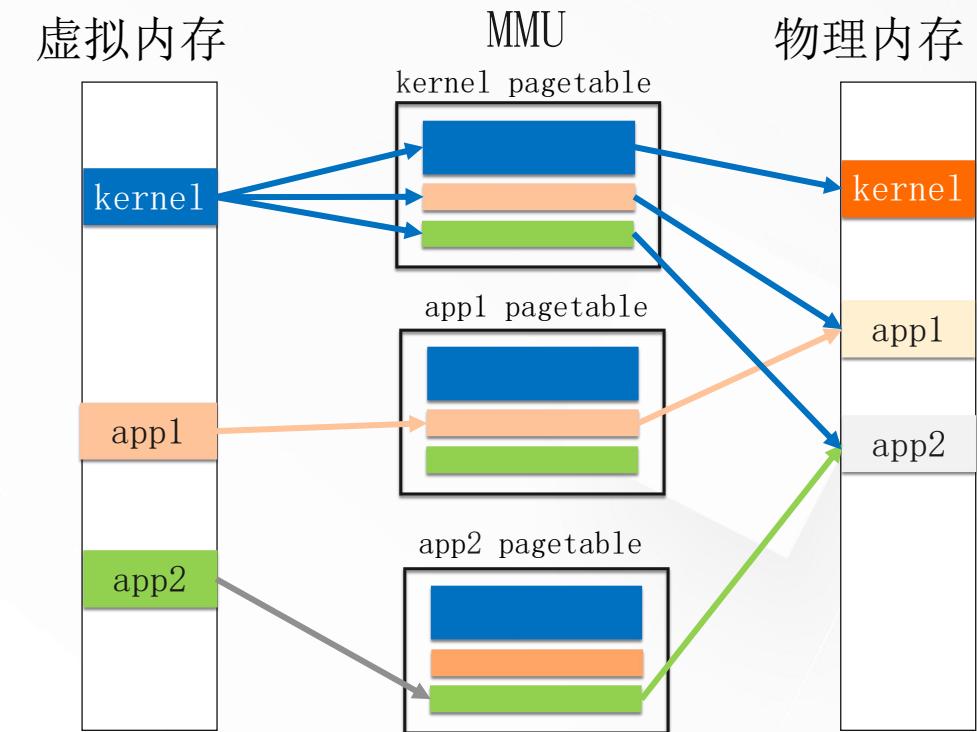


- 向上：应用层统一使用VFS类的驱动访问方式，与现有大量Linux生态兼容；
- 向下：驱动移植到AliOS Things时，可根据来源选择适合的方式：
 - 对接hal方式：适合RTOS驱动的移植，由OS的适配层进一步封装成对上接口。
 - 对接“xx子系统”方式：适合Linux驱动的移植，由OS抽象的驱动子系统负责管理。

多进程能力设计

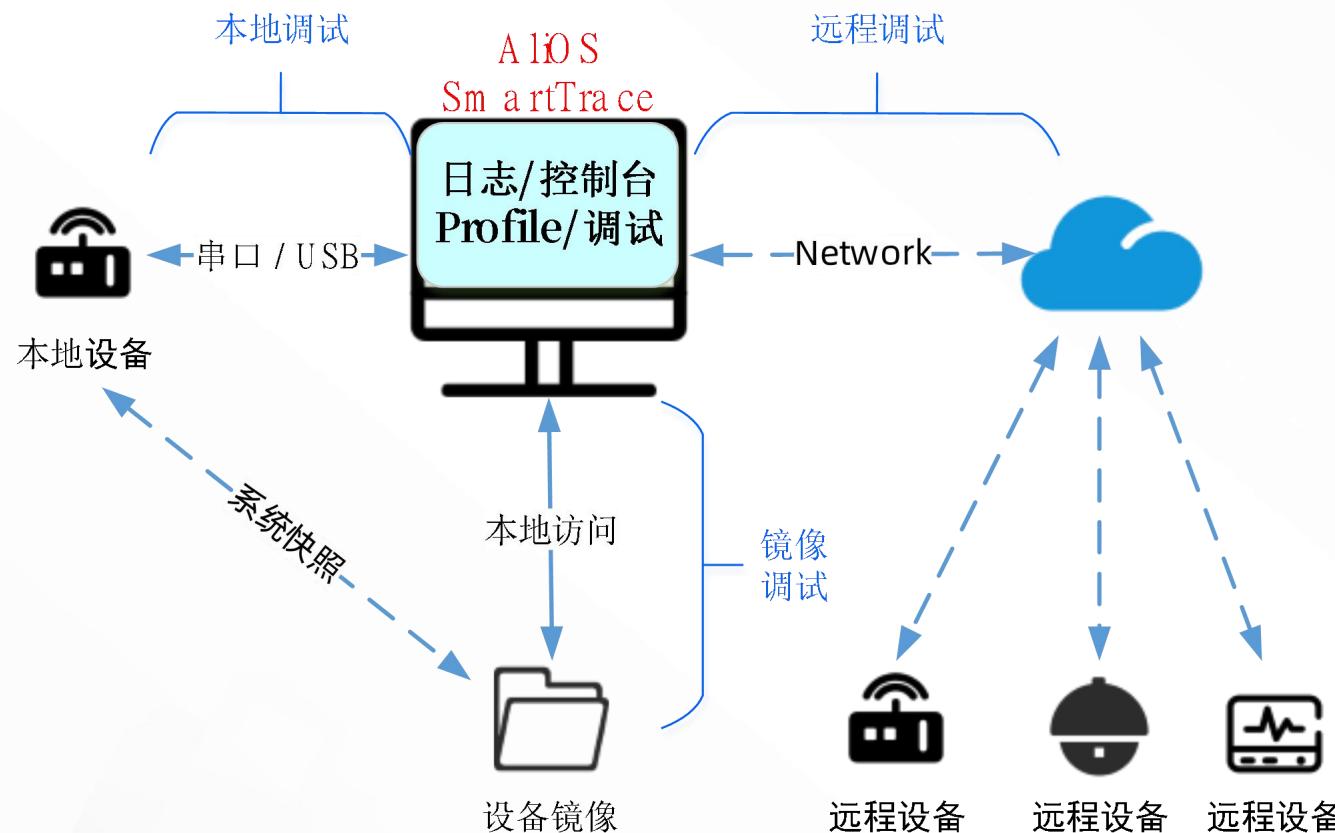


Cortex-M
不同进程配置不同的MPU region，实现内存隔离



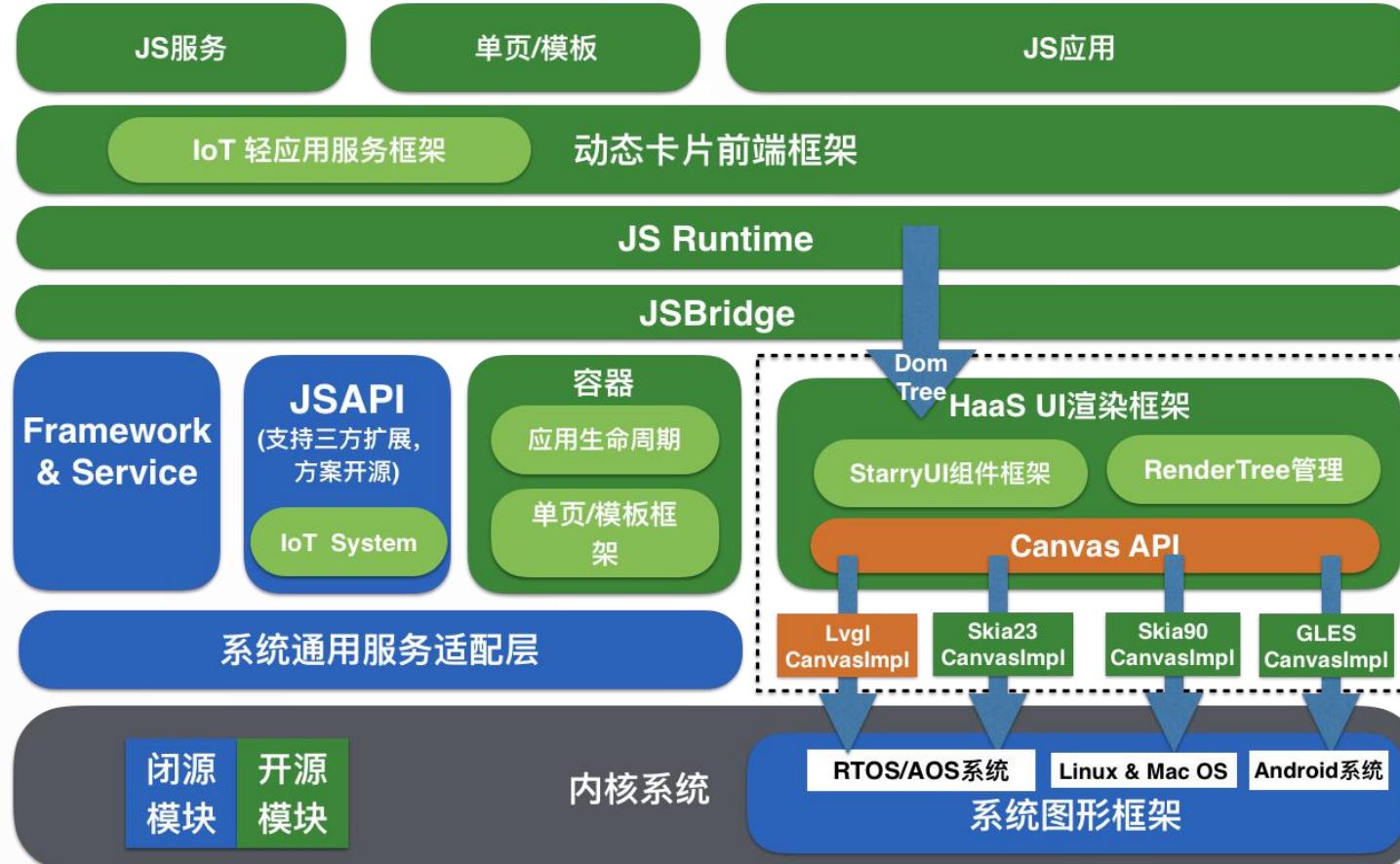
Cortex-A
利用MMU的虚拟内存与权限管理，实现内存隔离

诊断维测系统设计



- SmartTrace三种链接设备方式:
 - **本地**——串口、USB
 - **远端**——telnet/mqtt云端转发
 - **虚拟镜像**——来自设备的内存dump
- SmartTrace提供功能:
 - **智能CLI**——引入符号信息
 - **Profile**——OS性能分析
 - **智能日志**——关键字筛查、内容翻译等
 - **软件GDB**——不依赖JTAG的断点能力

UI交互显示设计



- HaaS UI
 - 支持JS开发的IoT轻应用
 - 自研渲染框架
 - 通过Canvas API可以快速完成渲染适配
- 资源占用
 - 内存范围1MB~16MB
 - Flash范围 0.7MB~16MB
 - 启动速度 < 1s
- 应用生态
 - 统一的JSAPI、组件、CSS样式
 - DSL基于Vue.js对外输出

AliOS Things操作系统应用案例



HaaS：加速AIoT中小开发者的创新平台

帮助AIoT中小开发者聚焦业务

低门槛快速组装软硬件积木

实现设备安全上云

加速AIoT创新迭代



定制化芯片



积木式软硬件



AliOS Things物联网操作系统



端到端方案

HaaS积木式标准硬件

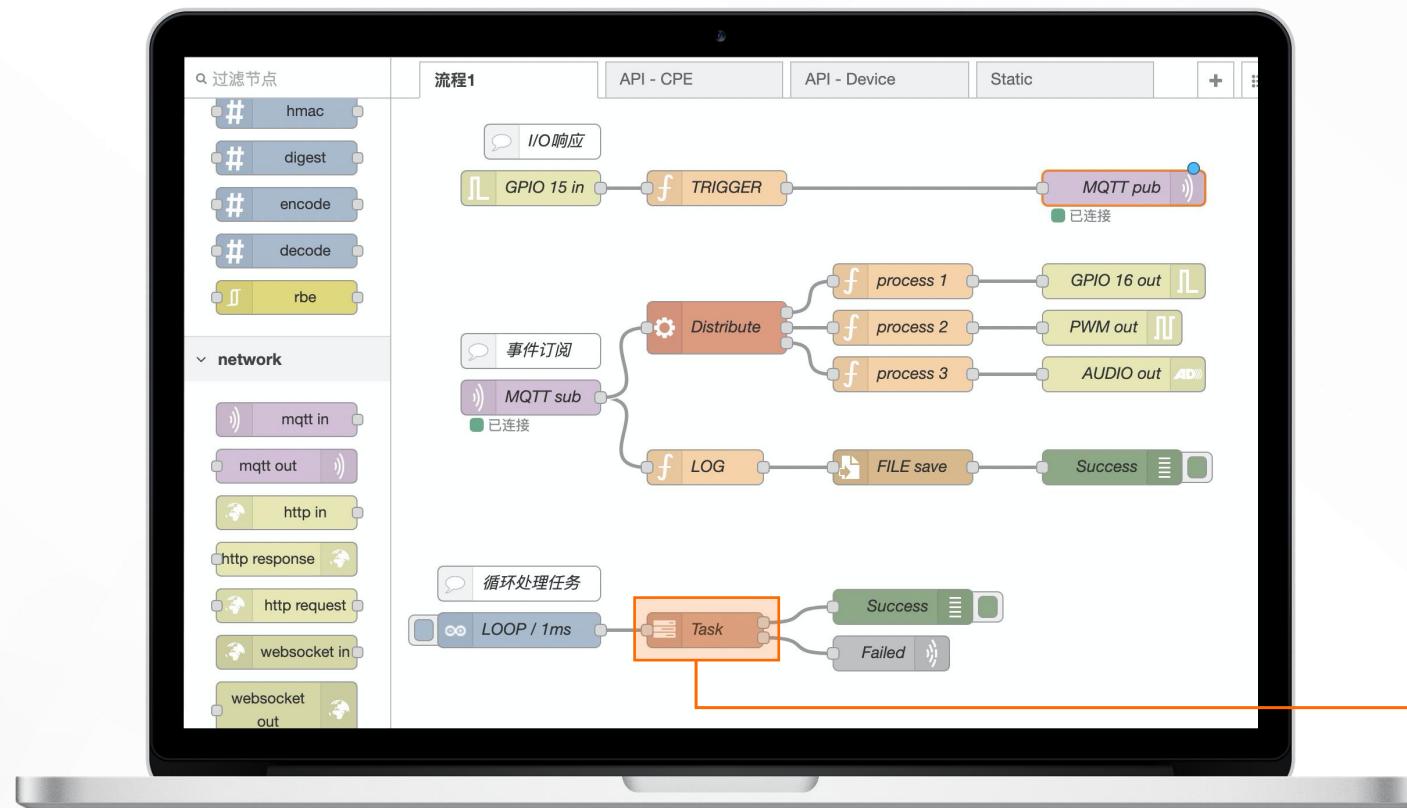


HaaS积木式丰富组件

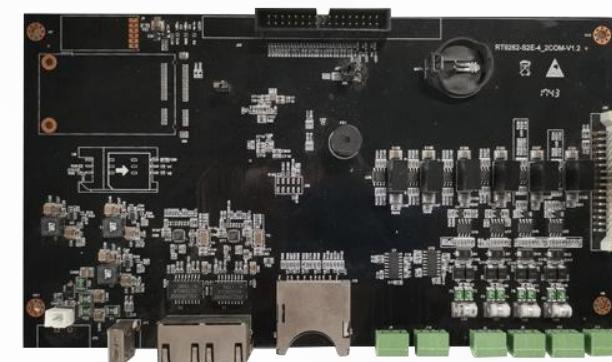
积木库

拖拽生成业务逻辑

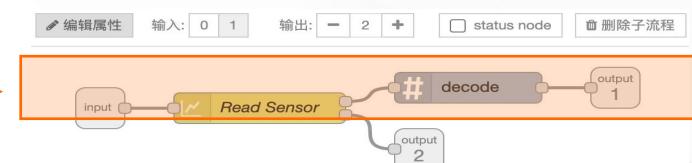
一键部署



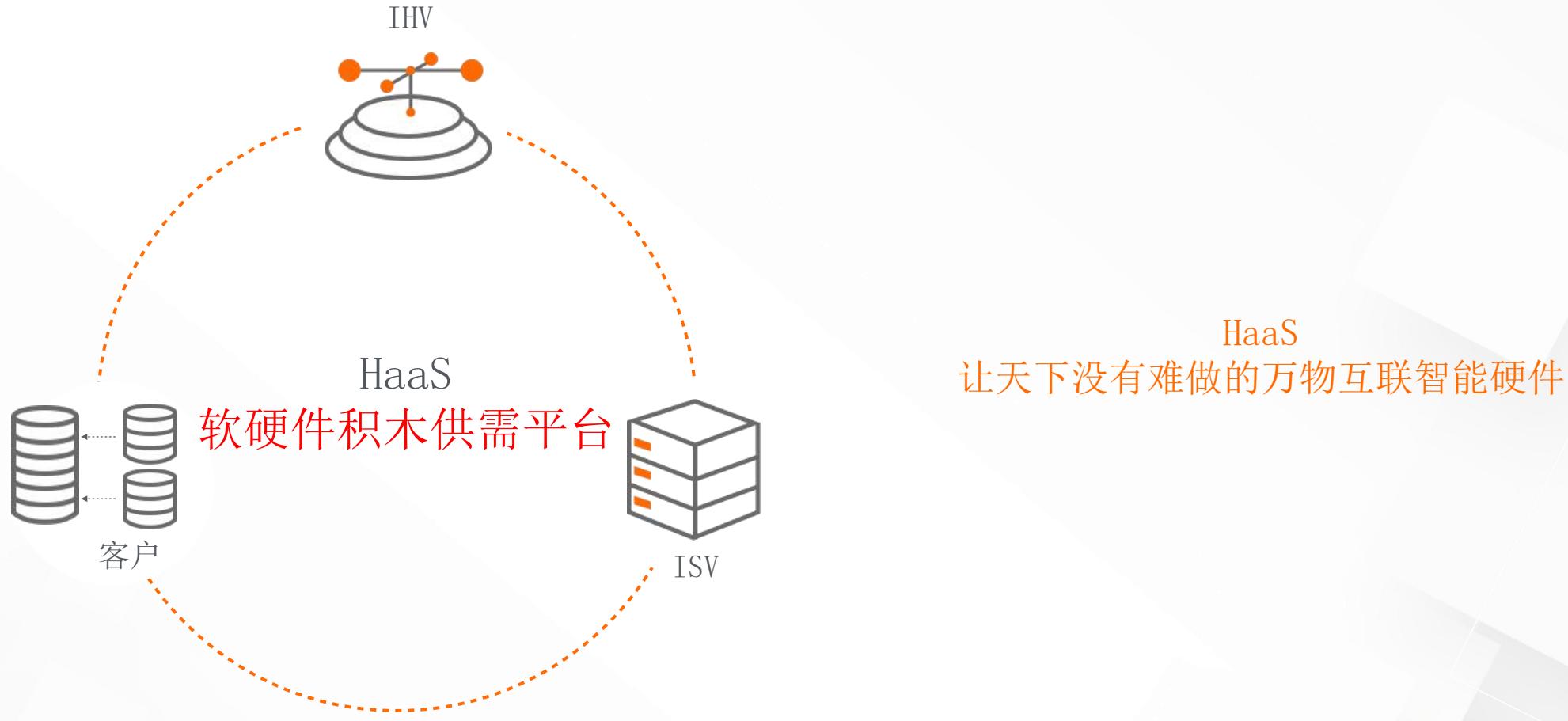
生成轻应用JS代码 一键热更新



子任务嵌套



HaaS平台生态合作共赢



阿里云

黄贵

阿里云数据库技术架构总负责人



黄贵，目前担任阿里云数据库技术架构总负责人，阿里巴巴资深技术专家，2008年加入阿里巴巴集团，从事分布式数据库系统开发十余年，参与PolarDB数据库核心技术研发工作，负责数据库整体架构规划与设计。在数据库国际顶级会议SIGMOD, VLDB, FAST发表过数篇论文。

主办方：

Boolan
高端IT咨询与教育平台

数据库在云原生时代的架构演进与技术特点



黄贵

阿里巴巴资深技术专家

2020/11

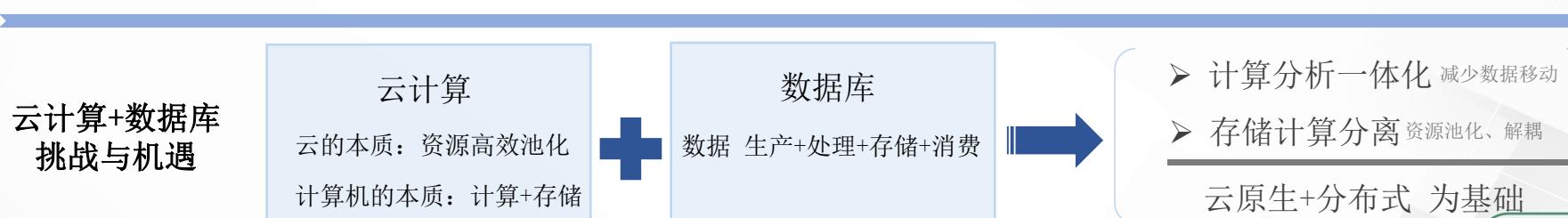
CPP-Summit

目录

- 01 云原生数据库发展时代背景**
- 02 数据库在云时代面临的主要挑战**
- 03 云原生数据库的架构演进**
- 04 云原生数据库的技术特点与实现**
- 05 未来发展方向**

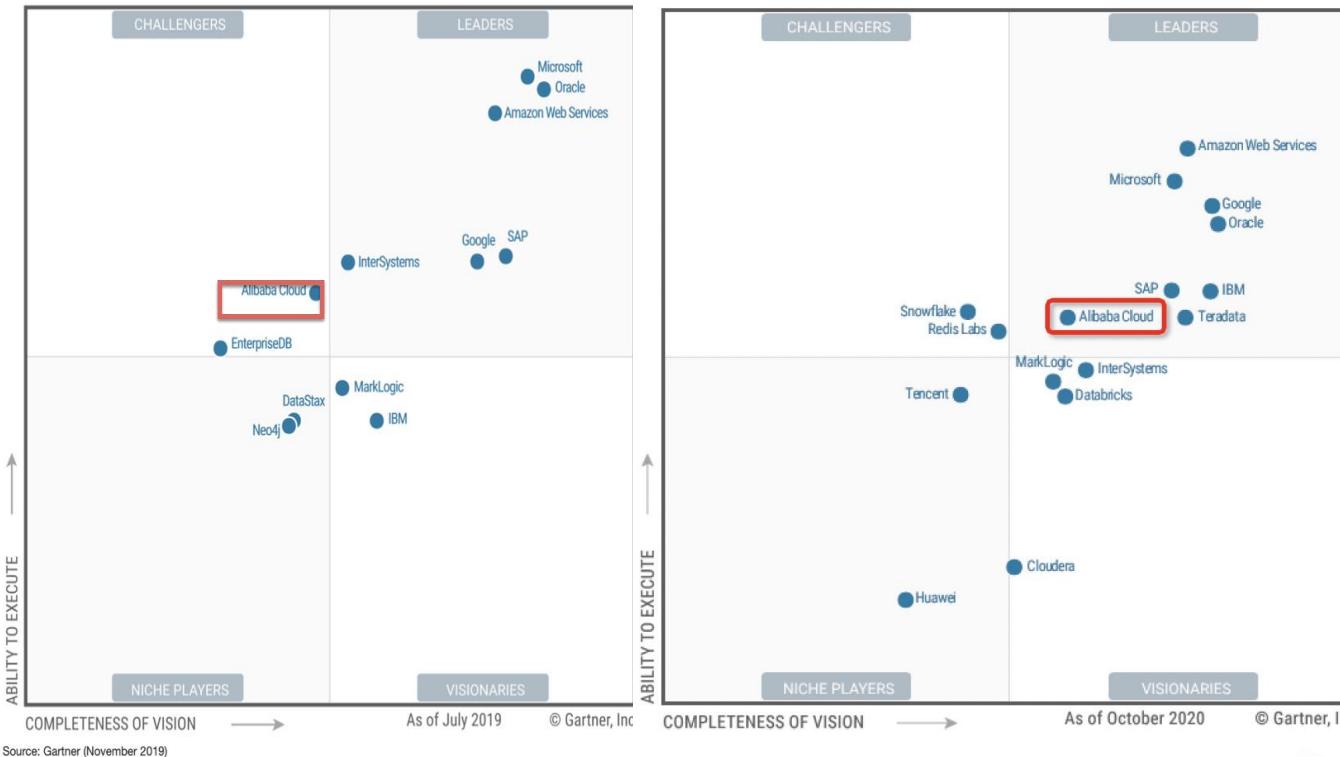
数据库系统演进

Evolution of Database System



云数据库正在成为一个新的趋势

- 在Forrester 2019年的研究中，已经有超过33%的全球基础设施商业决策者已经开始支持云原生数据库服务(DBaaS)。而在未来3到4年的时间内还会翻倍。
- 2018年全球云数据库收入占比22.75%，达到105亿美元，同比增长68%。预计到2022年，75%的数据库会被直接部署或向云上迁移。



Gartner 关系型数据库魔力象限报告中，云原生数据库(AWS, Google, Alibaba)在其中的重要性逐年提高，已经逼近甚至超过传统数据库巨头(Oracle, Microsoft)，且仍在高速发展。

数据来源：The Forrester Wave™: Database-As-A-Service, Q2 2019
 Gartner Magic Quadrant for Operational DBMS 2019

目录

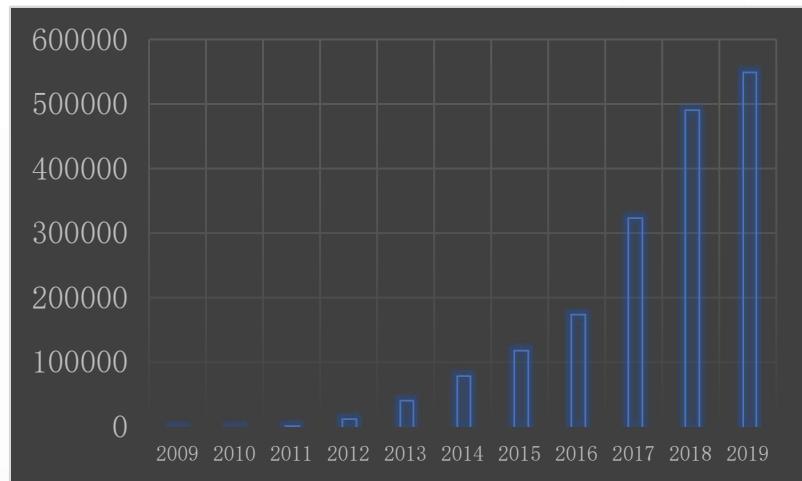
- 01 云原生数据库发展时代背景**
- 02 数据库在云时代面临的主要挑战**
- 03 云原生数据库的架构演进**
- 04 云原生数据库的技术特点与实现**
- 05 未来发展方向**

数据库在云计算时代面临新的挑战:高并发与弹性

阿里云

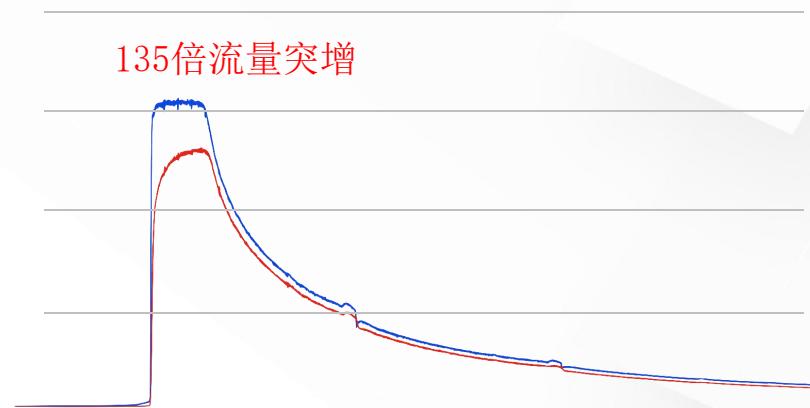
- 互联网业务的爆发式增长，对承载业务的数据带来**大规模的存储和访问需求**
- 各个行业广泛存在的流量瞬时突增，比如秒杀、双十一促销，**对系统造成极大冲击**

单机通过提升硬件无法满足对极高并发(**每秒数百万**)的访问需求



天猫双十一历年峰值交易笔数

流量较平日峰值突增**百倍**远超系统服务能力上限，依靠成倍加服务器浪费资源



2019双十一零点数据库请求曲线图

CPP-Summit

数据库在云计算时代面临新的挑战:高可用

阿里云

- 口 互联网带来的全球化特征，使得现代企业的客户遍及全球，服务对终端客户的全面触达，要求在线事务处理系统**不间断的提供可靠的服务**。

光缆被烧伤：京津宁骨干网中断

2017-04-01

继3月30日中国电信广州市从化区出现大面积光缆故障6个小时后，今日早上8点，京沪高速采育收费站北11公里的电缆井被烧，导致京津宁光缆中断，备用电路拥堵...

运营商反馈中断的光缆已于13:27左右彻底修复。

以下为[...]和[...]的公告，当然其他云服务商和互联网等用户也会受到影响，以下信息供各位参考~

[...]对用户的通知：

您好！收到骨干网反馈：京津宁光缆中断，导致部分线路流量激增。受此影响，北京电信至浙江、江苏、黑龙江、四川、辽宁、贵州、福建等部分地区线路会出现延迟增大及丢包现象，目前我司已调整自建骨干网（调整自建骨干网可恢复120.92.X.X段EIP访问，第三方BGP、CDN及KS3业务需等运营商处理后恢复），请贵司关注业务情况。我司将持续关注事态进展，待有新进展时随时更新通报。回复TD退订【...】

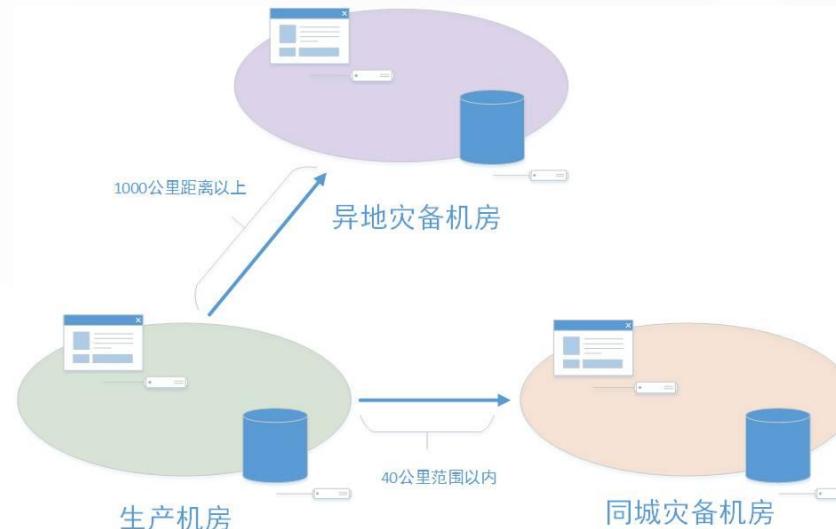
[...]：北京电信网络异常通告

[...] [运营商] [故障通告]

故障简述：监控发现于2017年04月01日08:05左右，北京电信访问互联网出现不稳定（包括[...]华北2地域），09:44左右不稳定程度加剧，目前 [...] 已经向运营商报障，会持续配合运营商处理。

故障原因：电信运营商网络链路不稳定。

有任何问题，可随时通过工单或服务电话 [...] 联系反馈。

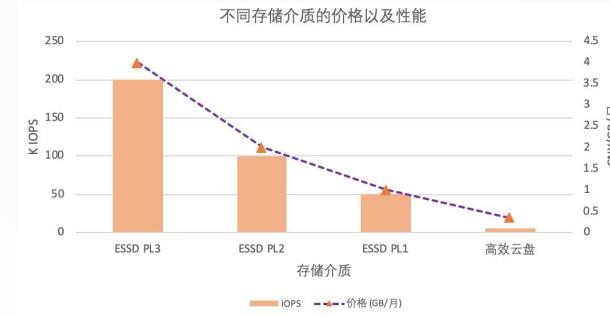
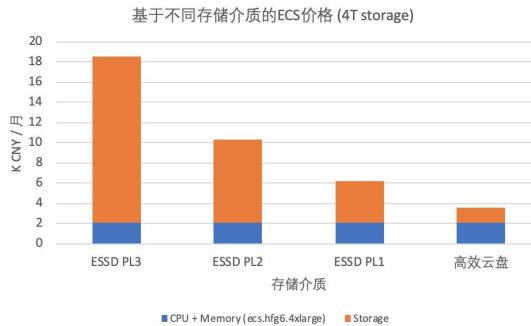


CPP-Summit

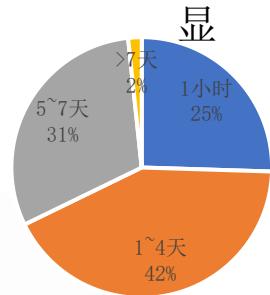
数据库在云计算时代面临新的挑战:高可用

阿里云

问题：业务增长带来的数据持续增长^[1]，而使用高速存储介质带来巨大成本压力

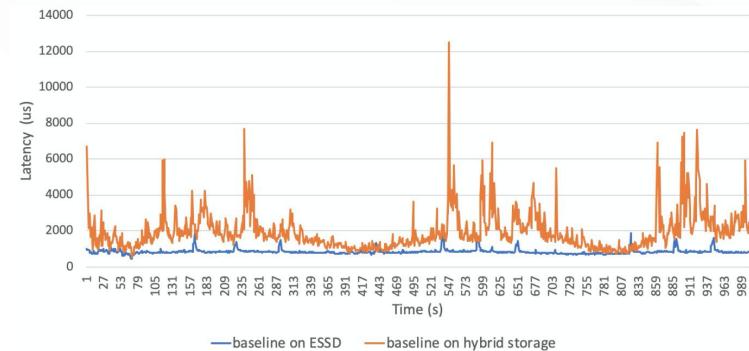


大数据量下数据的热度区隔明显



交易数据生成后，7天后仍旧会被访问的概率只有1.69%

直接压缩或是直接使用廉价存储会使性能大幅下降



[1] 数据来源：<https://clouddba.alibaba-inc.com/>

目录

- 01 云原生数据库发展时代背景**
- 02 数据库在云时代面临的主要挑战**
- 03 云原生数据库的架构演进**
- 04 云原生数据库的技术特点与实现**
- 05 未来发展方向**

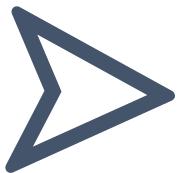
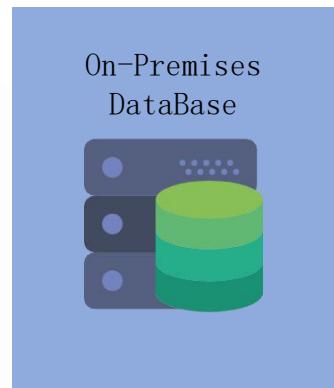
什么是云原生数据库

阿里云

A Cloud-Native database is a sort of database service which is used to build, deployed and delivered through cloud platforms

简单来说，云原生数据库就是一种通过云平台构建、部署和交付的数据库服务。

数据库由一个系统演变成为一个服务，即数据库作为一种服务(Database-as-a-Service)



Secure



On-Demand

Self-
Service

Scalable
Elasticit
y

CPP-Summit

云原生数据库的主要特点

On-Demand: 云数据库利用云平台提供的数据库服务，随时可用，可以根据业务需求购买适合的规格和资源，用完即可释放。避免了传统数据库购买，部署，运维以及更新的生命周期管理。

Security: 云数据库服务比大多数人想象的更安全。云平台有完善的安全体系，经过考验的网络防御措施，并且持续不断升级最新的技术来保证商业数据和敏感信息的数据安全。

Elasticity: 弹性可谓云数据库服务最重要的特性，云平台维护规模化的资源池，可以根据应用的工作负载特征进行资源的弹性伸缩，维持对业务负载的最佳资源配置。

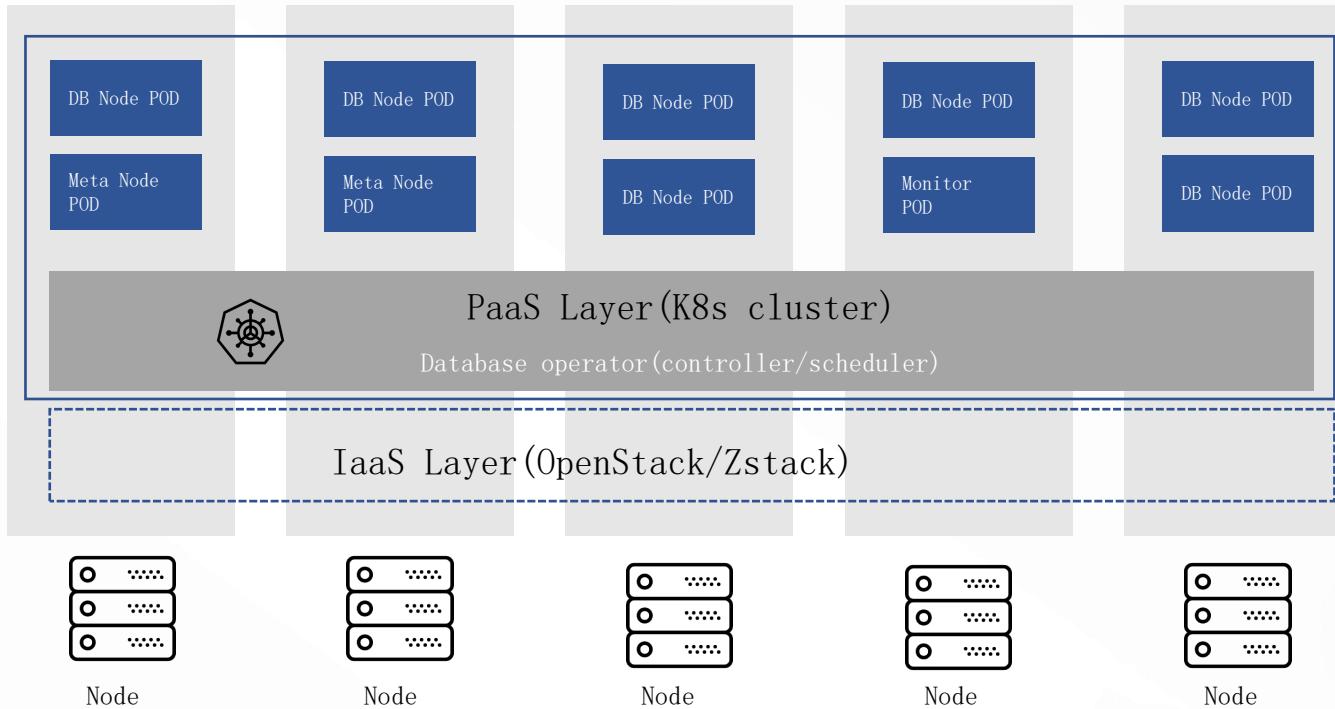
High Availability: 云数据库一般都利用复制副本的内建高可用机制来避免单点失效，利用云平台多个可用区的资源，实现跨多个地区的服务高可用能力，即使遇到灾难性故障也保证服务一直在线。

Cost Effective: 云数据库使用多租户/Serverless技术极小化使用成本，大幅降低数据库使用门槛，因为强大的即时弹性能力，无需像传统数据库一样事先根据业务发展趋势预留资源，仅仅需要用户为使用到的资源付费。

DataBase on Cloud Platform

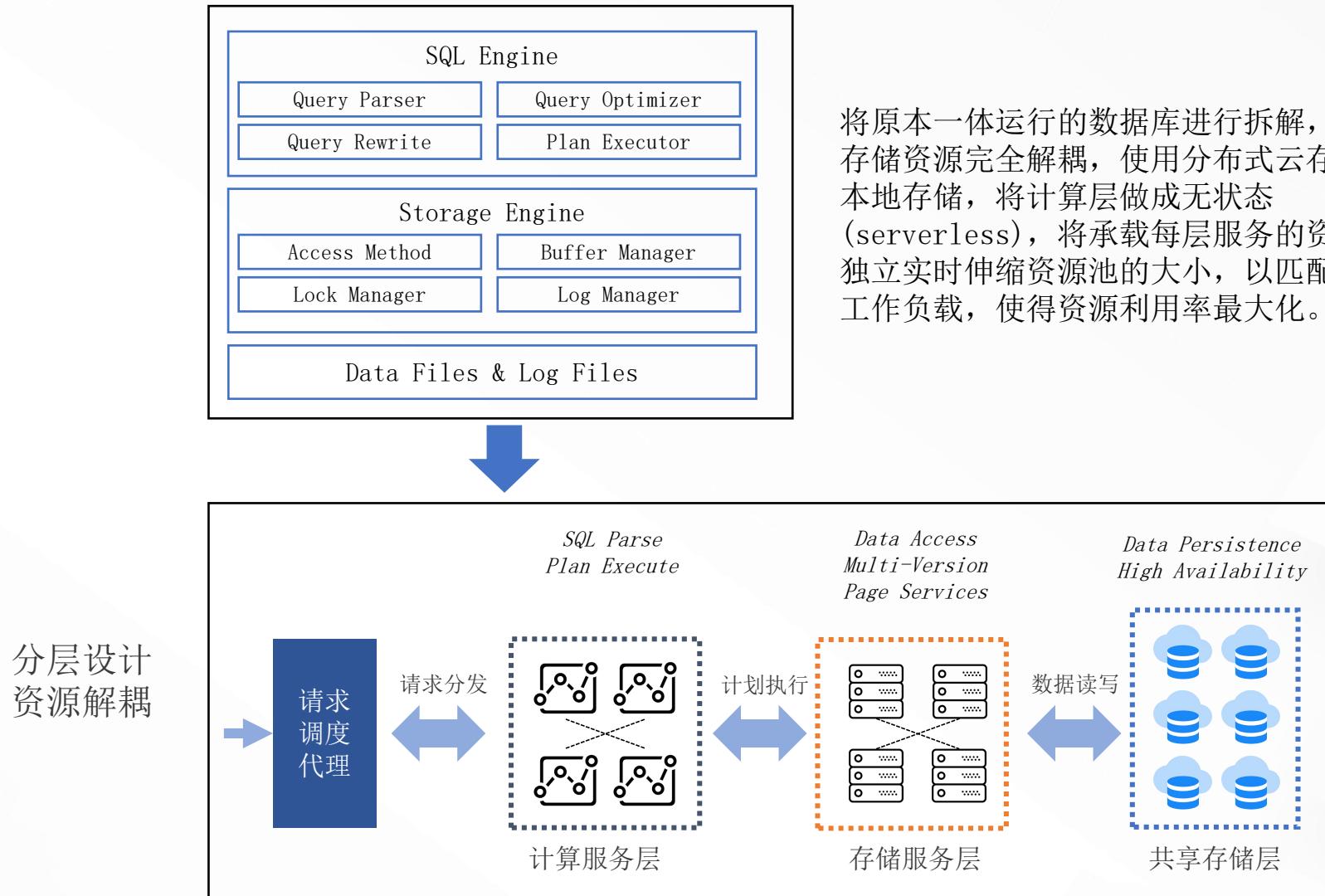
阿里云

把数据库搬到云上，利用云平台弹性调度，资源隔离的能力，提供开箱即用的数据库服务。
一般云数据库服务提供数种固定规格，业务变化需要匹配对应的规格升降配，难以原地实时弹性。



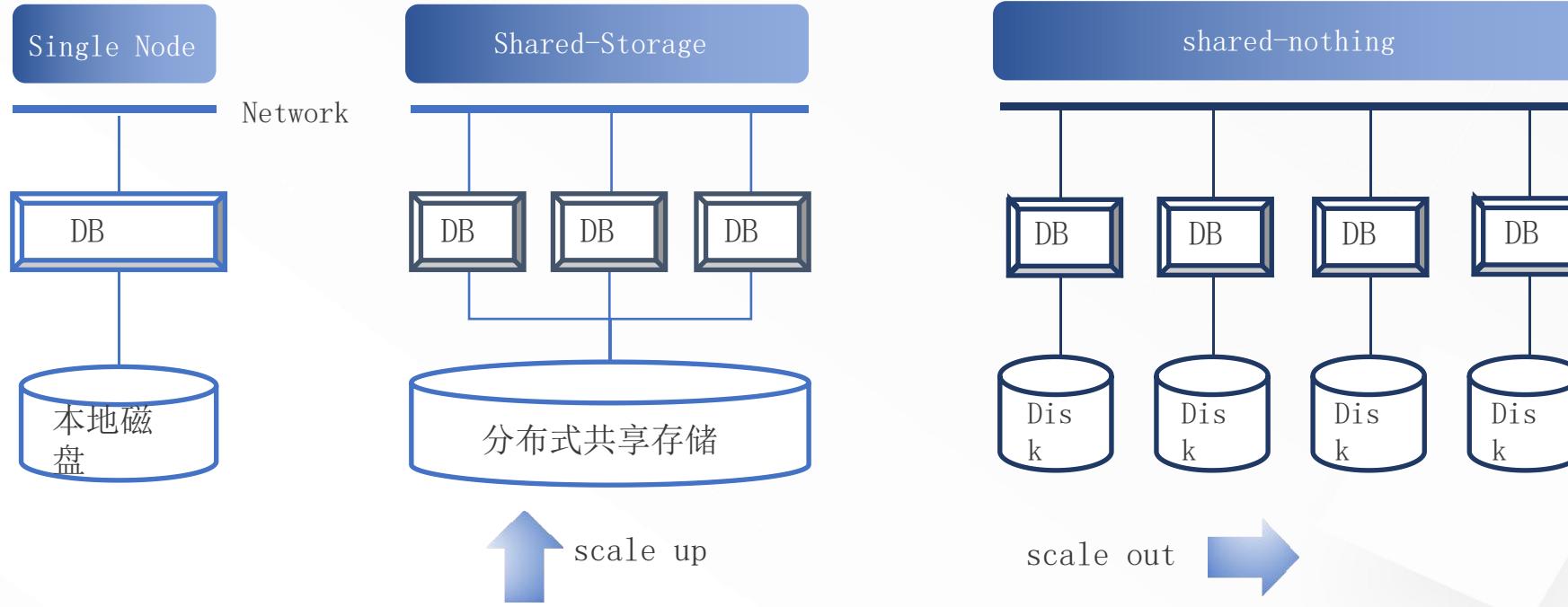
云原生数据库的架构演进

阿里云



CPP-Summit

Shared-Storage and Shared-nothing



- ◆ 完全兼容单机数据库
- ◆ 事务处理简单
- ◆ 计算存储独立扩展
- ◆ 扩展性受限

- ◆ 良好的水平线性扩展能力
- ◆ 多副本高可用, 不依赖共享存储
- ◆ 难以独立扩展计算和存储
- ◆ 分布式查询以及分布式事务

目录

01 云原生数据库发展时代背景

02 数据库在云时代面临的主要挑战

03 云原生数据库的架构演进

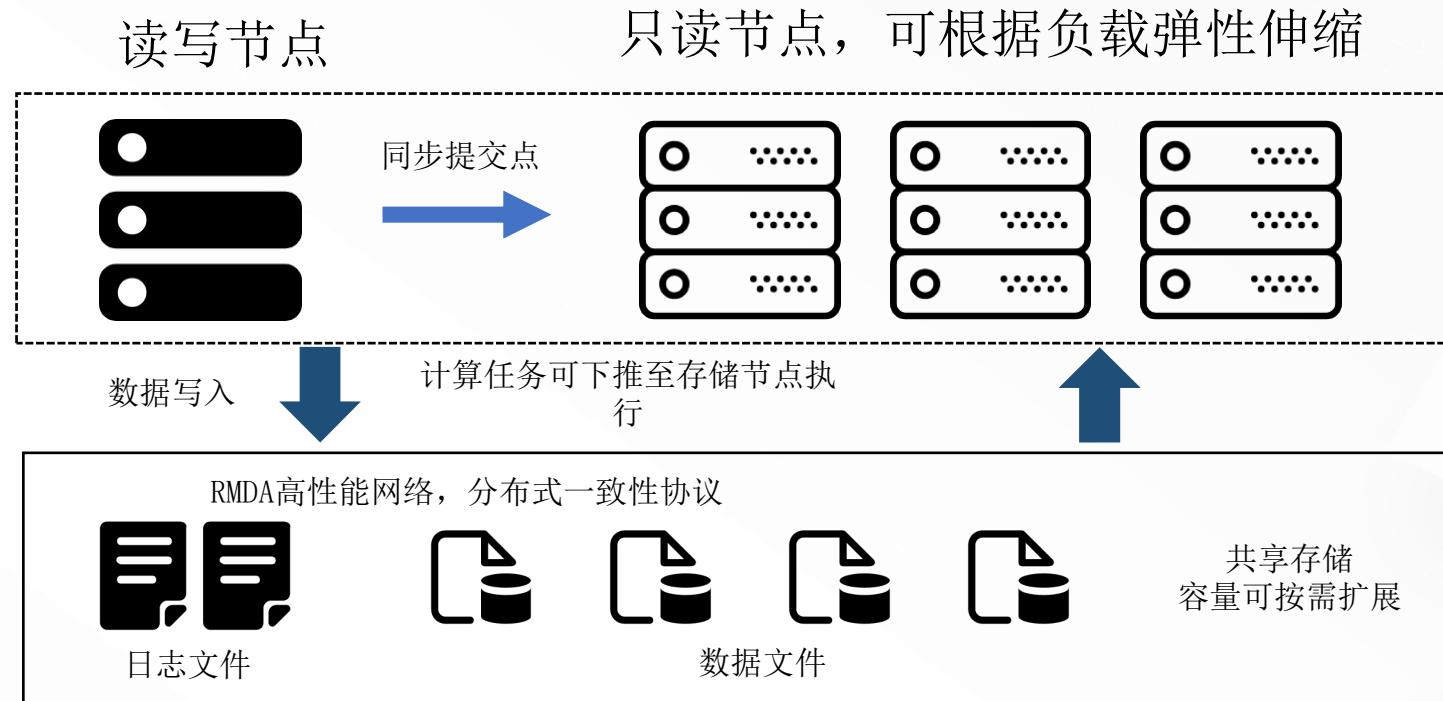


04 云原生数据库的技术特点与实现

05 未来发展方向

利用共享存储实现计算的扩展:一写多读能力

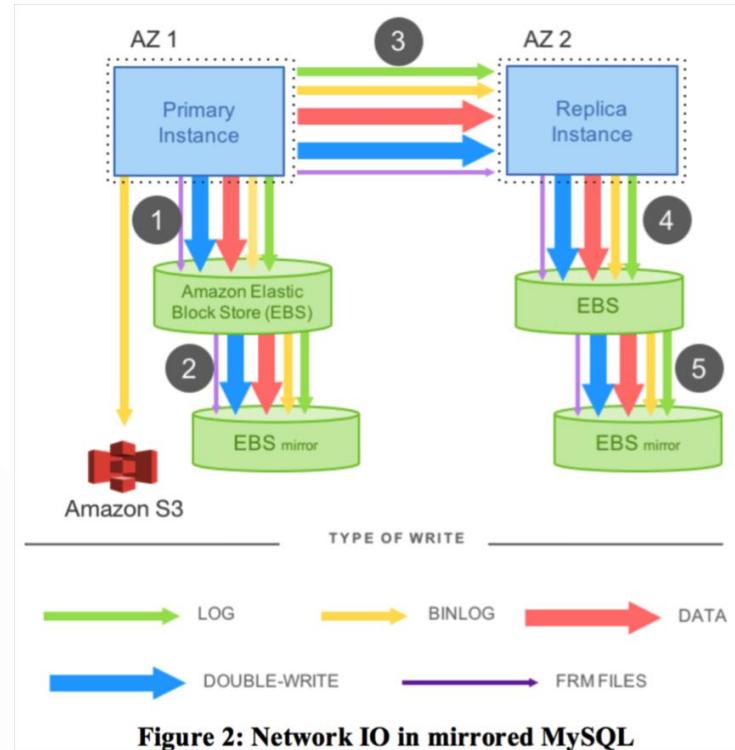
阿里云



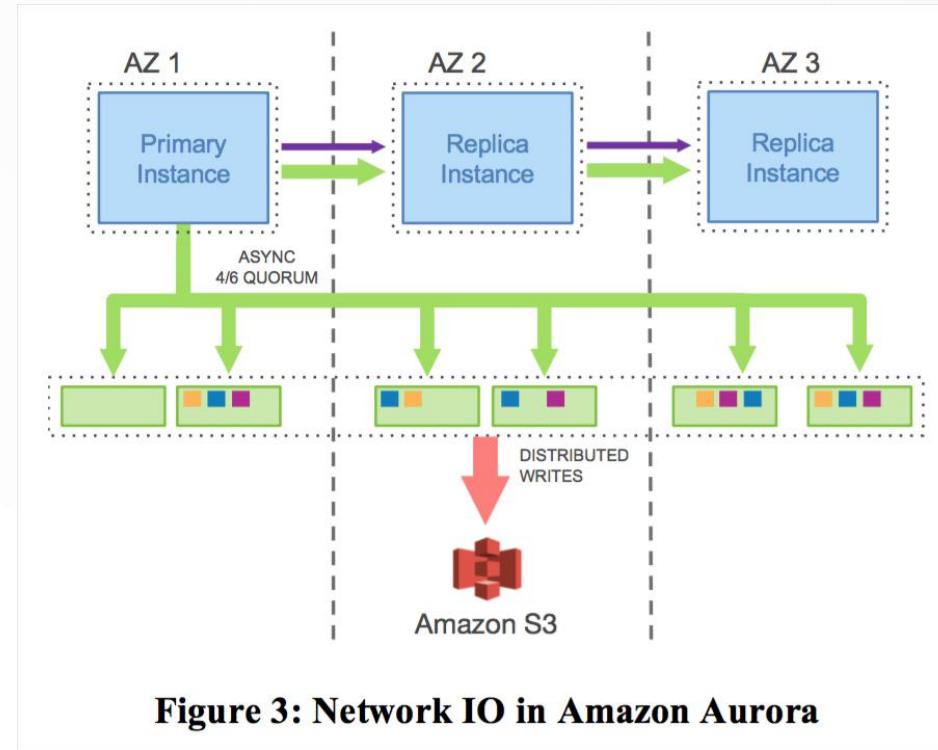
典型共享存储架构数据库:AWS Aurora

阿里云

传统数据库写放大问题

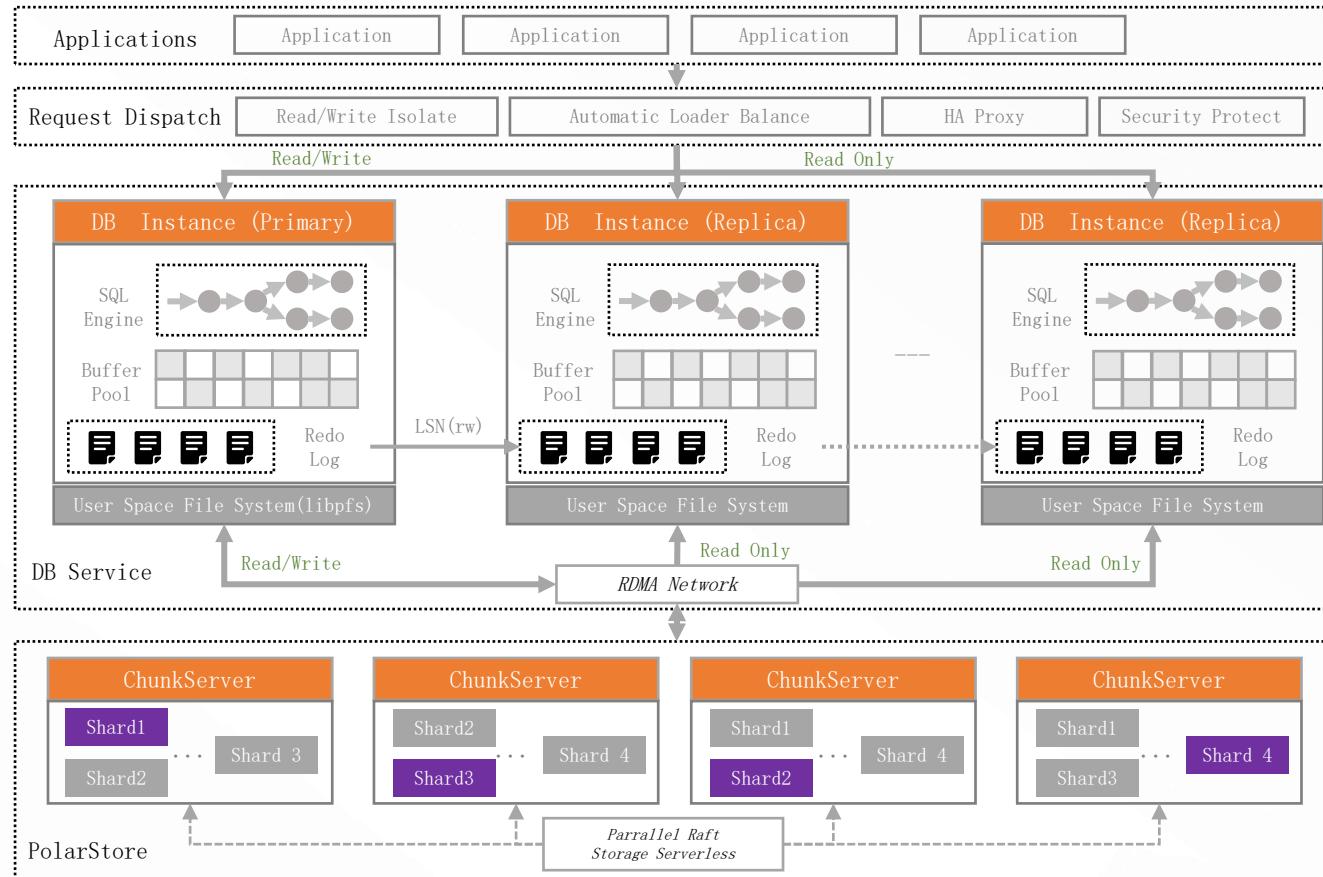


日志处理下放到存储层



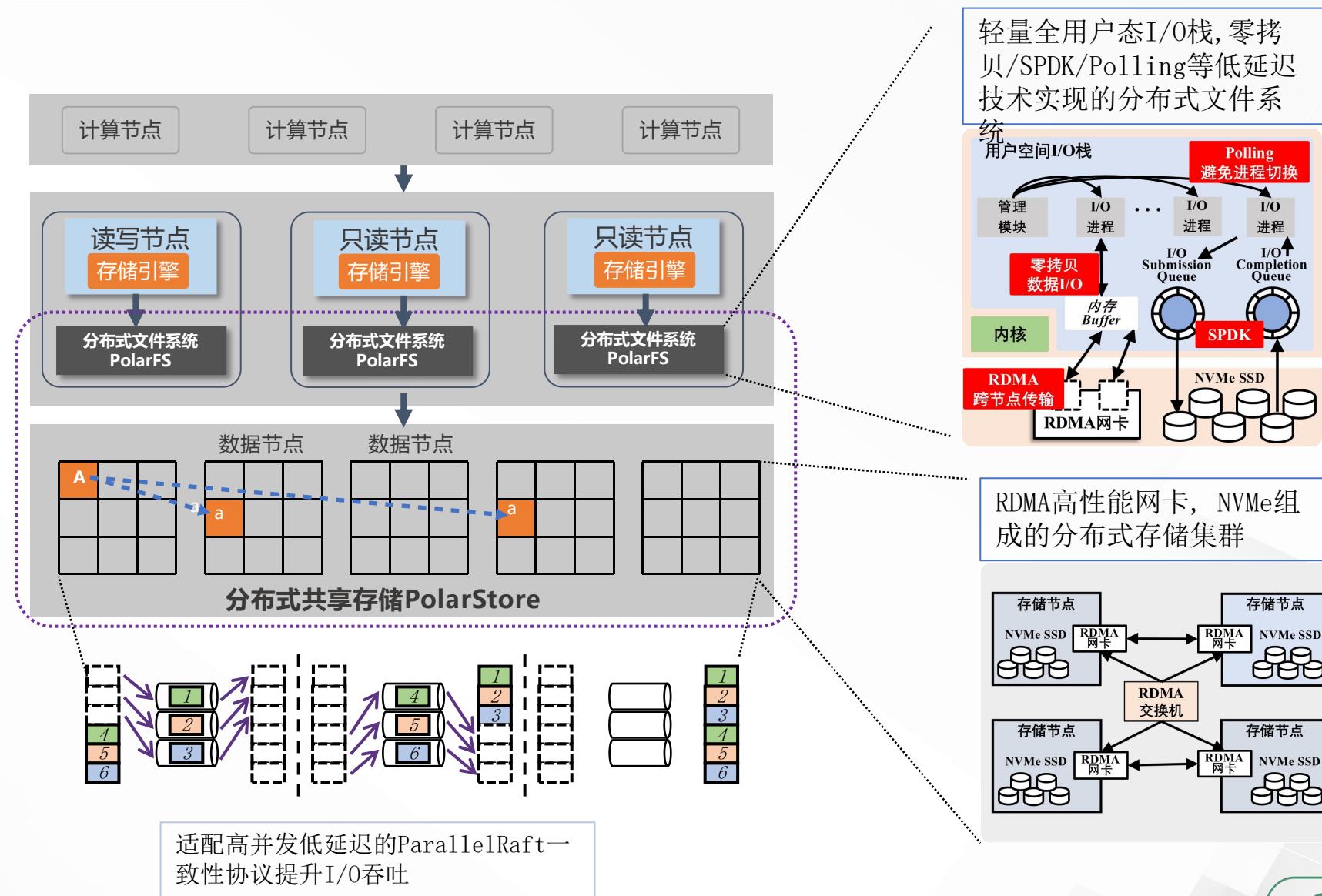
PolarDB:高性能网络共享存储

- ◆ 利用RDMA高性能网络构建的分布式共享存储
- ◆ Parallel Raft 实现多副本数据一致
- ◆ 用户态I/O栈与计算下推减少延时



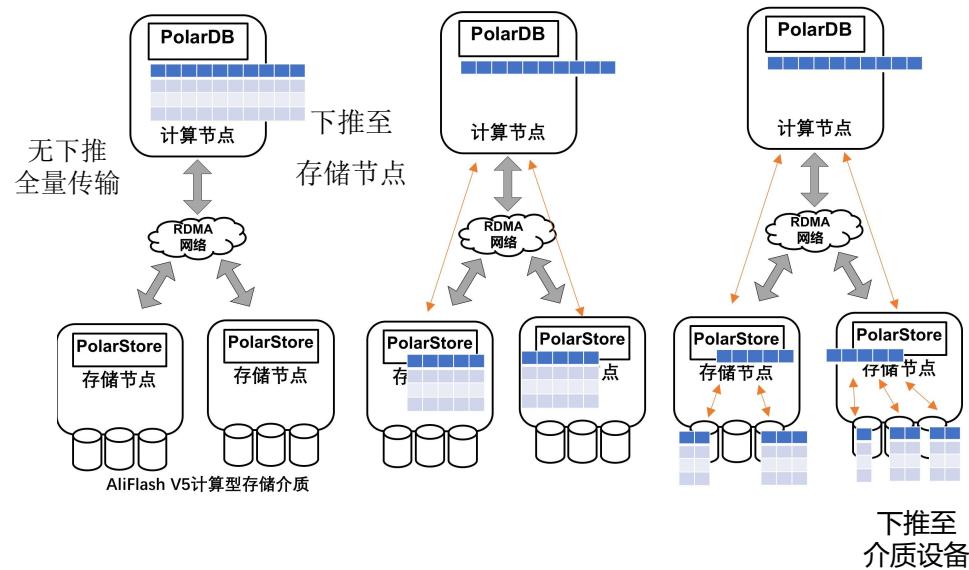
共享存储系统高可用与延时优化

阿里云

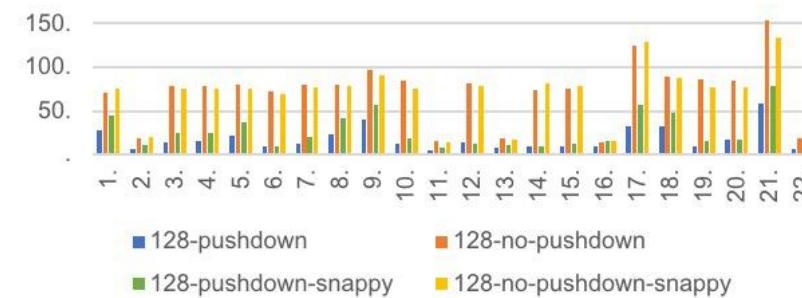


近存储计算

计算全路径打通：计算引擎→存储引擎→文件系统→存储节点→计算型介质
计算操作在最贴近数据的地方进行，减少I/O数据量传输，减少延时。

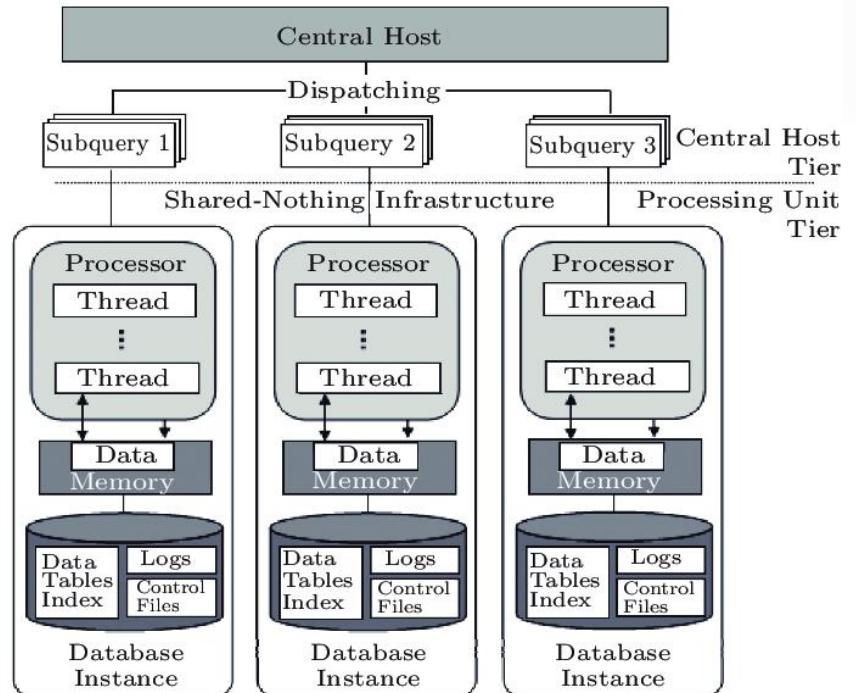


计算下推 TPC-H 100G 测试

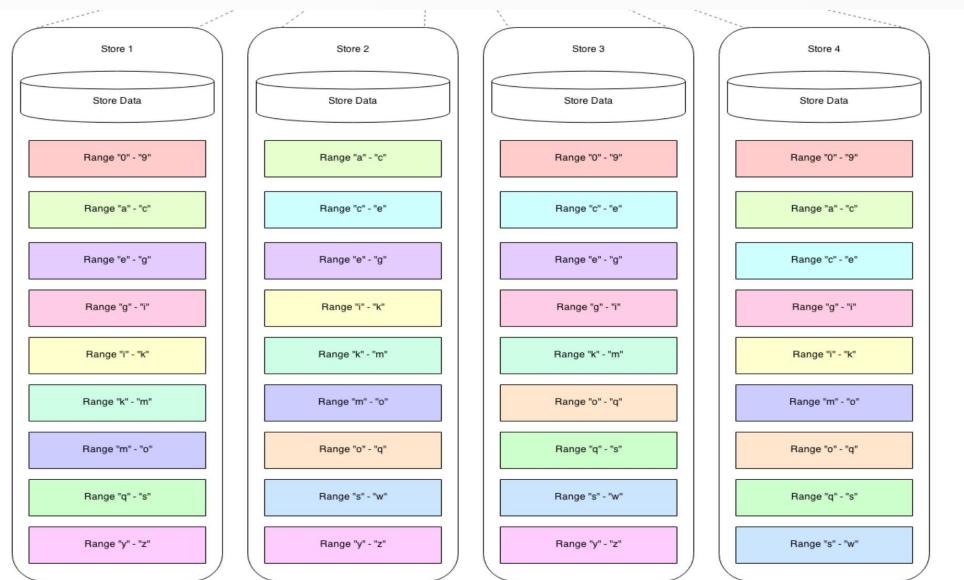


数据分片以及高可用复制组

数据分片(sharding)与查询路由

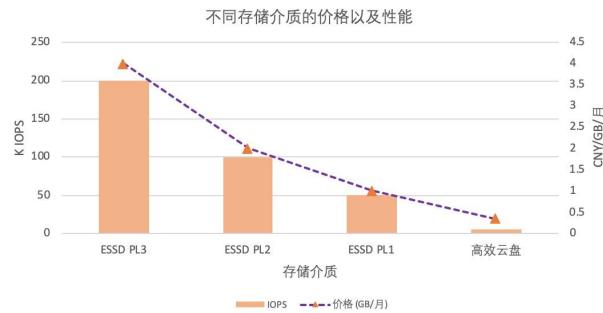


数据分片算法：
hash/range/interval...
多副本高可用复制组

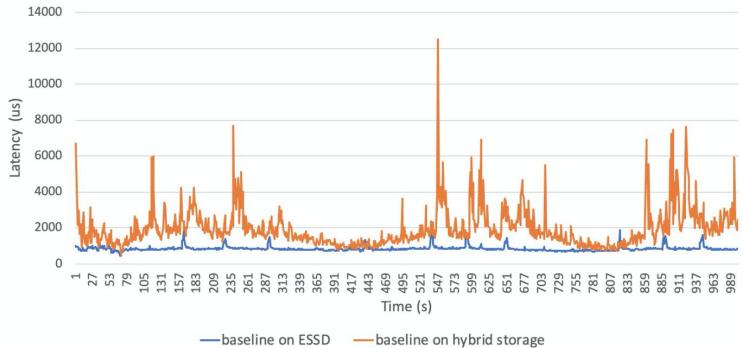


新型存储引擎降低存储成本

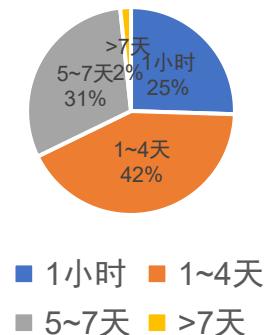
存储介质与成本密切相关



直接压缩或是直接使用廉价存储会使性能大幅下降



大数据量下
数据的热度区隔明显

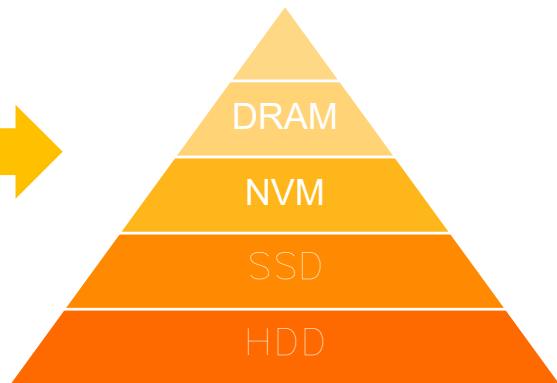


交易数据生成后，7天后仍旧
会被访问的概率只有1.69%

不同热度的数据
适合不同的存储方式



新型存储设备
带来更为丰富的存储层次



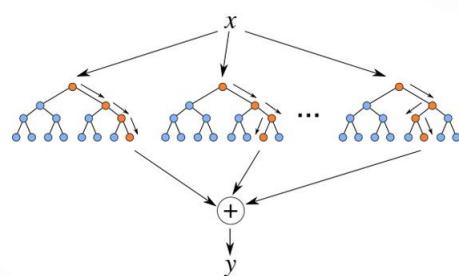
Memory-Storage
Hierarchy

根据数据访问冷热特征分层的存储引擎

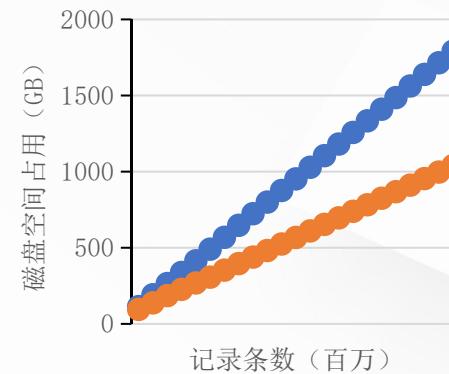
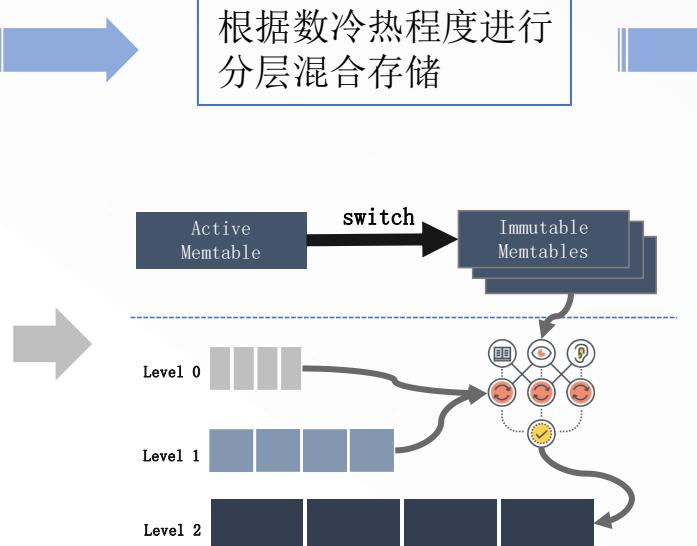
根据数据访问特征，
智能识别数据冷热程度

根据数冷热程度进行
分层混合存储

冷数据类型格式
感知编码压缩



基于可解释轻量级的决策树模型的冷数据预测算法



LSM 架构的存储引擎，因为写入友好，便于压缩的特性得到了广泛关注，其分层结构也非常适合根据冷热数据进行分层，进一步降低综合成本。参考：[X-Engine: An Optimized Storage Engine for Large-scale E-commerce Transaction Processing \(SIGMOD' 2019\)](#)

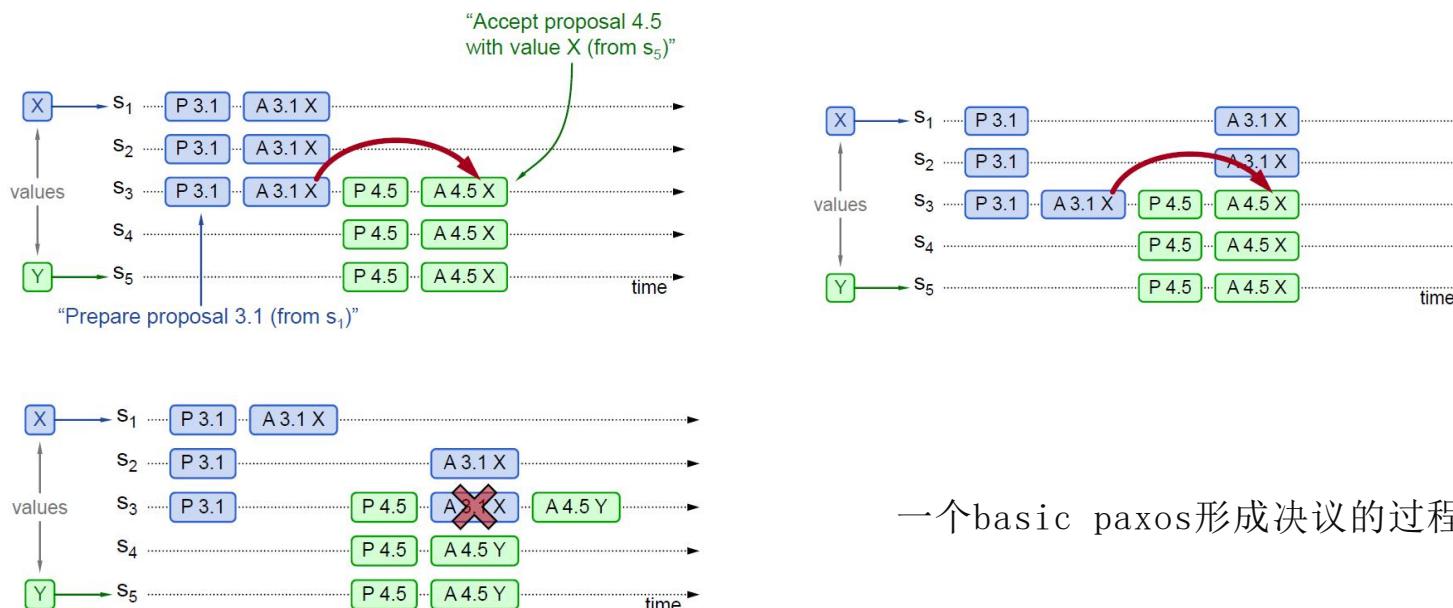
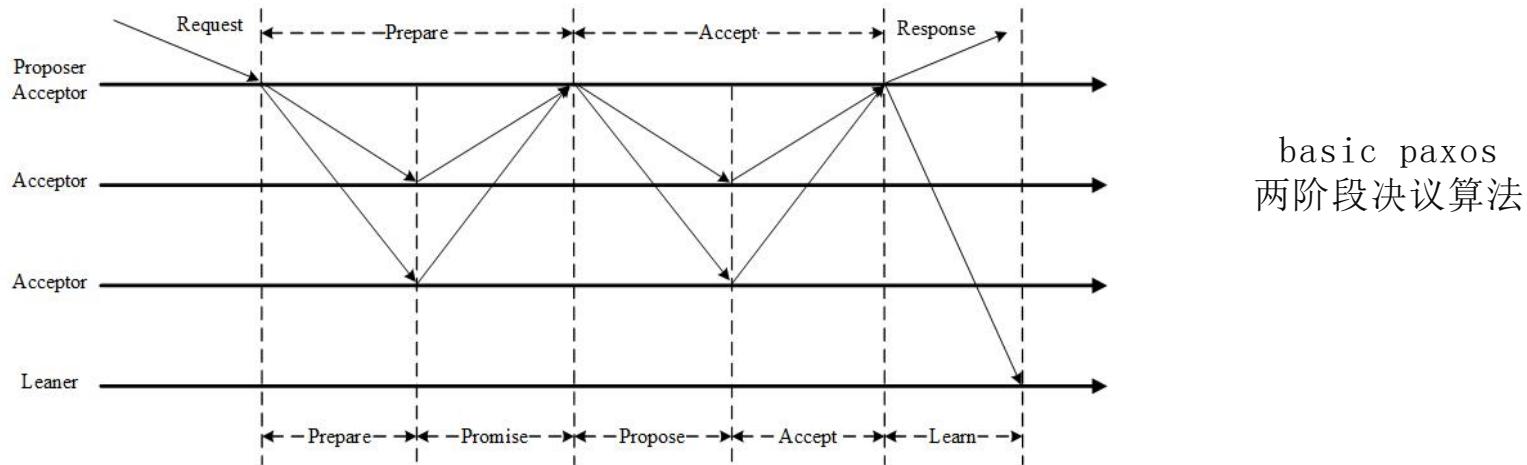
高可用： CAP理论证明了分布式系统不可能同时满足Consistency, Availability, Partition Tolerance，实际应用领域中，Partition一定会发生，C/A是不可兼得的，但这不意味着我们需要放弃其中一个，因为实际系统并不追求同时保证100%一致性和可用性。TP数据库对数据强一致性的要求是无法妥协的，因此在保证强一致的前提下，能够保证多大程度的可用性便是分布式数据库需要着重思考的问题。

综述：如何在分布式数据库系统中对一致性进行取舍：[Consistency Tradeoffs in Modern Distributed Database System Design](#) (DJ Abadi, Computer, 2012)

Paxos：分布式一致性协议是保证分布式数据库系统数据强一致的关键，而每种分布式一致性协议或是算法最终都是Paxos及其变体。[\(原始论文：The Part-Time Parliament, ACM Transactions on Computer Systems 1998\)](#)。Paxos以其晦涩著称，不易理解，所以Leslie Lamport后来又写了简单版本([Paxos Made Simple](#))；即便如此，工程实现也殊为不易，[为此](#) David Mazières写了([Paxos Made Practical](#))探索了工程实现。还有其他的一些一致性算法比如([Viewstamped Replication, ZooKeeper的zab协议](#))实际上都可以视为Paxos变体，可以选读。

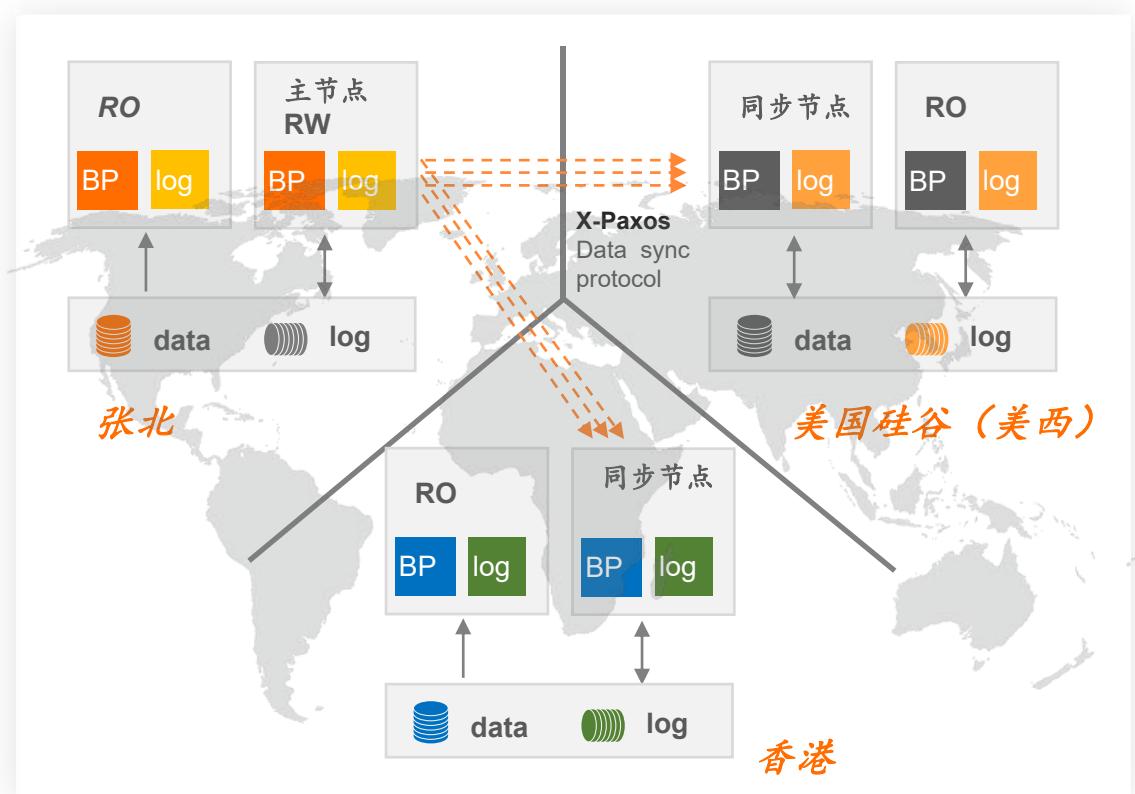
Raft：Paxos由于其难于理解在实际工业系统中落地的并不多，Diego Ongaro设计了一套等价于Paxos但是通过一些增强约束大大提升了Paxos的可理解性，把Paxos工业应用向前推进了一大步。[\(In Search of an Understandable Consensus Algorithm, USENIX ATC ‘2014\)](#)

Paxos算法：多个分布式参与者达成一致



跨域高可用能力

PolarDB 全球数据库 (Global Database)



全球部署

Global Deployment

数据跨地域同步，提供全球跨地域的容灾能力

就近读加速

Accelerate by Reading the Nearest Node

读操作就近读取数据，适合不同地域读多写少的场景

多通道物理复制

Multi-Channel Physical Replication

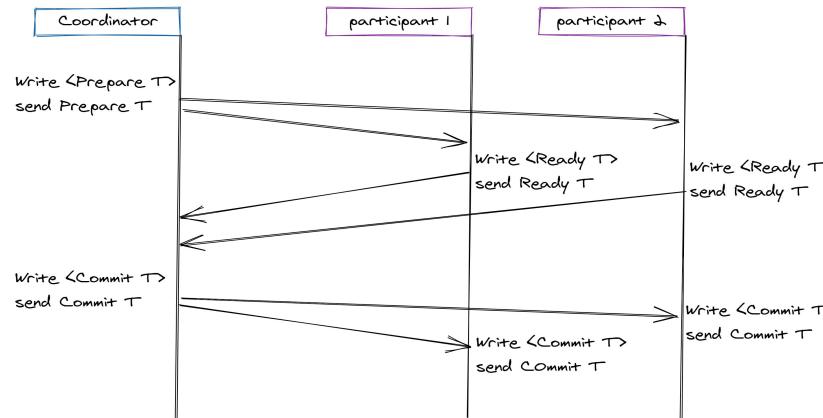
提供数据跨地域的高速同步，大压力场景下全球同步延迟确保在2秒以内

多点跨地域写

Write to Multi Zones and Regions

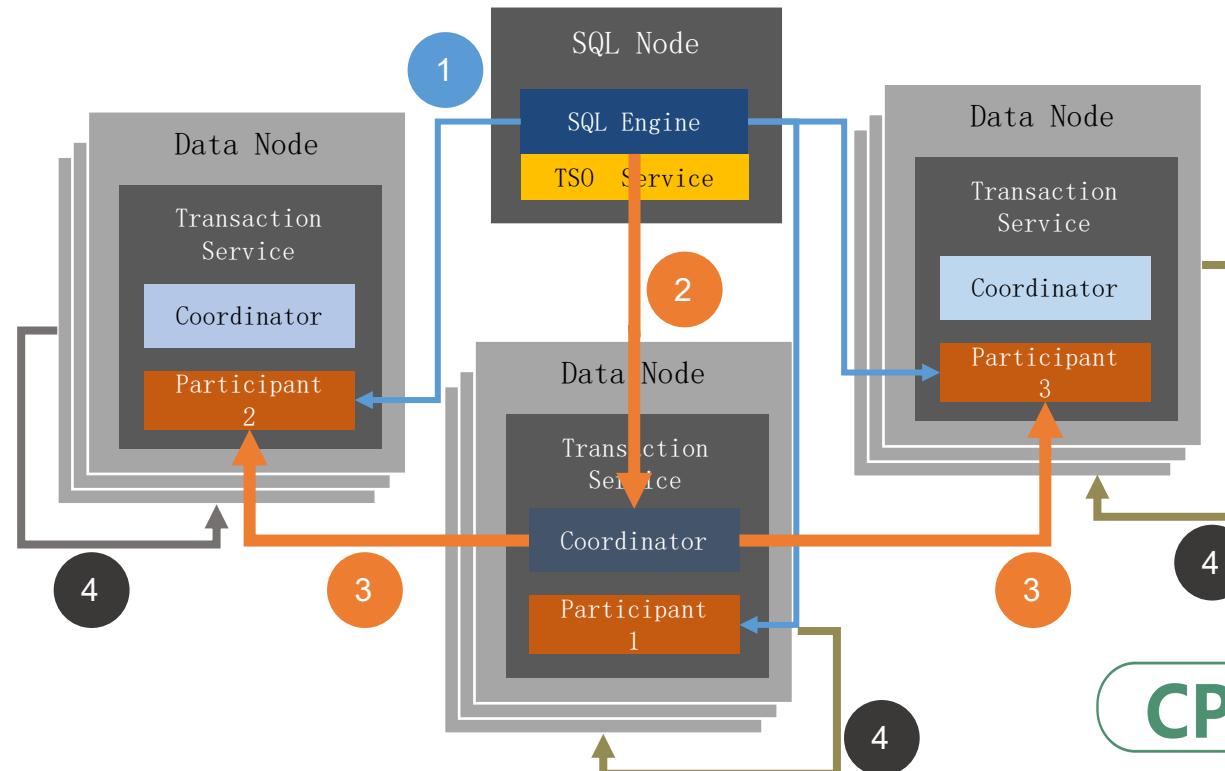
提供多点跨地域写功能，提供业务的多地部署能力

分布式事务处理



两阶段提交(2 Phase Commit)
标准流程

1. 事务调度者SQL Node拿取事务号(TSO/HLC timestamp)，执行事务中的每条语句，在相关的participants所在的Server，DataNode执行事务语句，开始事务的读写阶段，对事务进行冲突检查，对操作的数据上锁，将事务写入transaction buffer
2. 事务提交，选定一个participant作为coordinator，执行commit；
3. coordinator开始发起两阶段事务，对每个participants发送prepare，每个participant执行事务写入，写prepare日志。
4. 在paxos group写入多数派
5. 发送commit；每个part写commit日志并同步到paxos group多数派
6. 返回SQL Node事务处理结果，进入事务清理阶段。



分布式事务处理技术

分布式事务：分布式数据库的扩展性必须需要数据分区，随之而来的后果就是必须处理跨数据分区的分布式事务。

综述：An Evaluation of Distributed Concurrency Control (VLDB' 2017)。

The End of a Myth: Distributed Transactions Can Scale (VLDB' 2017)

Percolator: Google在分布式Key-value store(BigTable)上利用TSO和行原子性操作实现分布式事务处理协议。Large-scale Incremental Processing Using Distributed Transactions and Notifications (OSDI' 2010)

Spanner: 利用原子钟和Commit Wait机制实现分布式事务，保证外部一致性。Spanner: Google's Globally Distributed Database (TOCS' 2013)

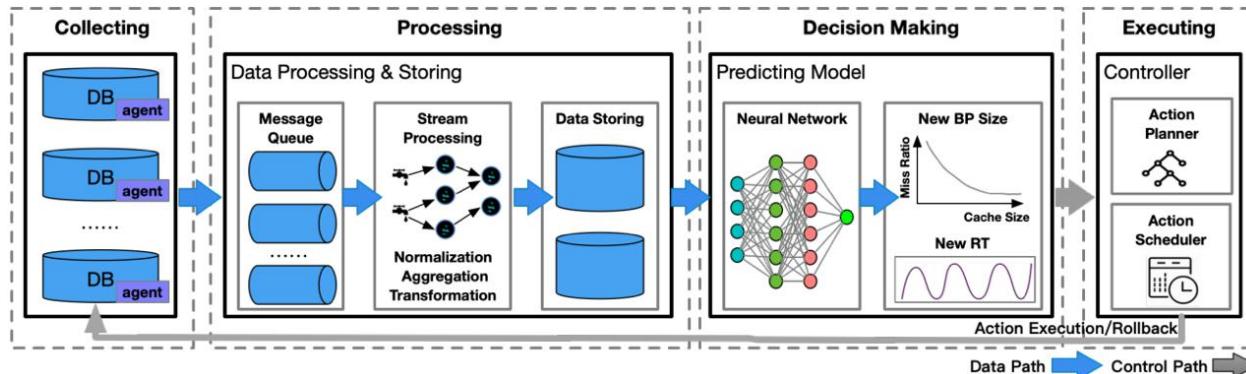
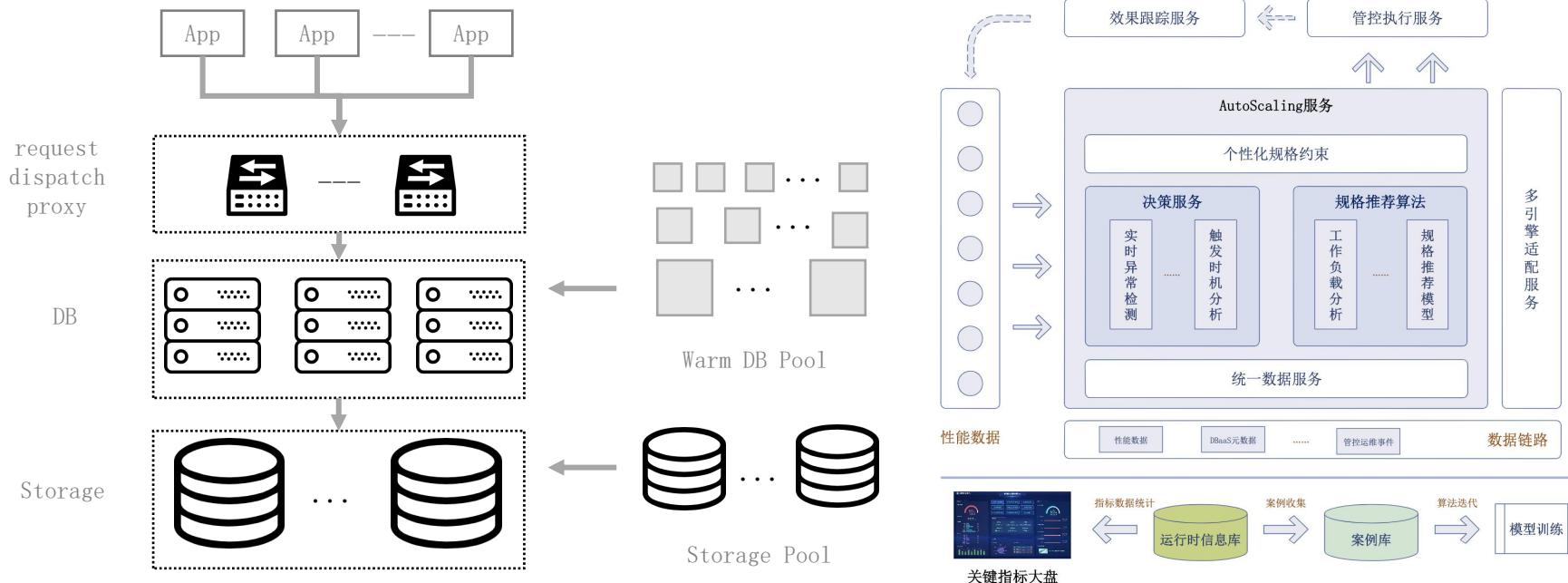
C \angle L \sqcup = \emptyset : 确定性事务排序批量执行机制。Calvin: Fast Distributed Transactions for Partitioned Database Systems (SIGMOD' 2012)

HLC: 使用TSO(全局时间戳)对分布式事务进行定序，TSO会成为局部单点，影响扩展性，混合逻辑时钟根据事务相关性生成时间戳，解决扩展性问题。

Logical Physical Clocks and Consistent Snapshots in Globally Distributed Databases

CockroachDB: The Resilient Geo-Distributed SQL Database (SIGMOD' 2020)

Serverless autoscaling



使用机器学习算法根据负载模型进行规格推荐，定义弹性模型进行实时扩缩容

目录

- 01 云原生数据库发展时代背景**
- 02 数据库在云时代面临的主要挑战**
- 03 云原生数据库的架构演进**
- 04 云原生数据库的技术特点与实现**
- 05 未来发展方向**

云原生+分布式：数据库数仓一体化

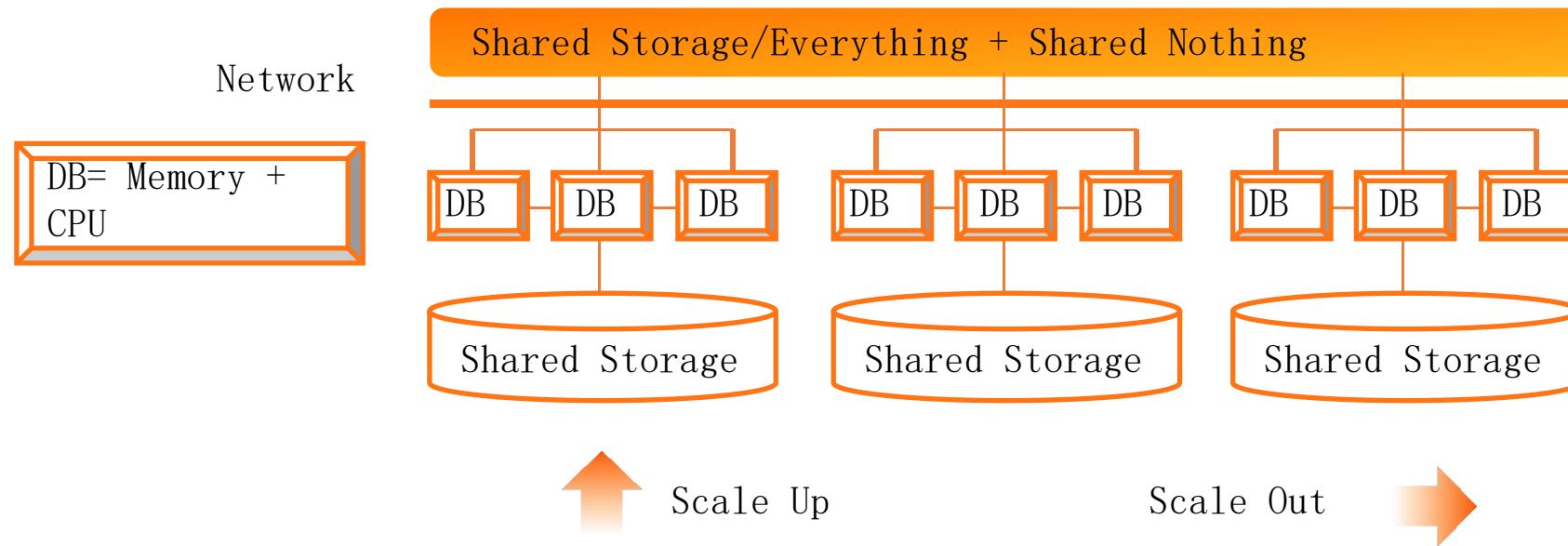
阿里云

Next Generation Enterprise-level Database System: Cloud Native + Distributed

弹性

高可用

水平拓展

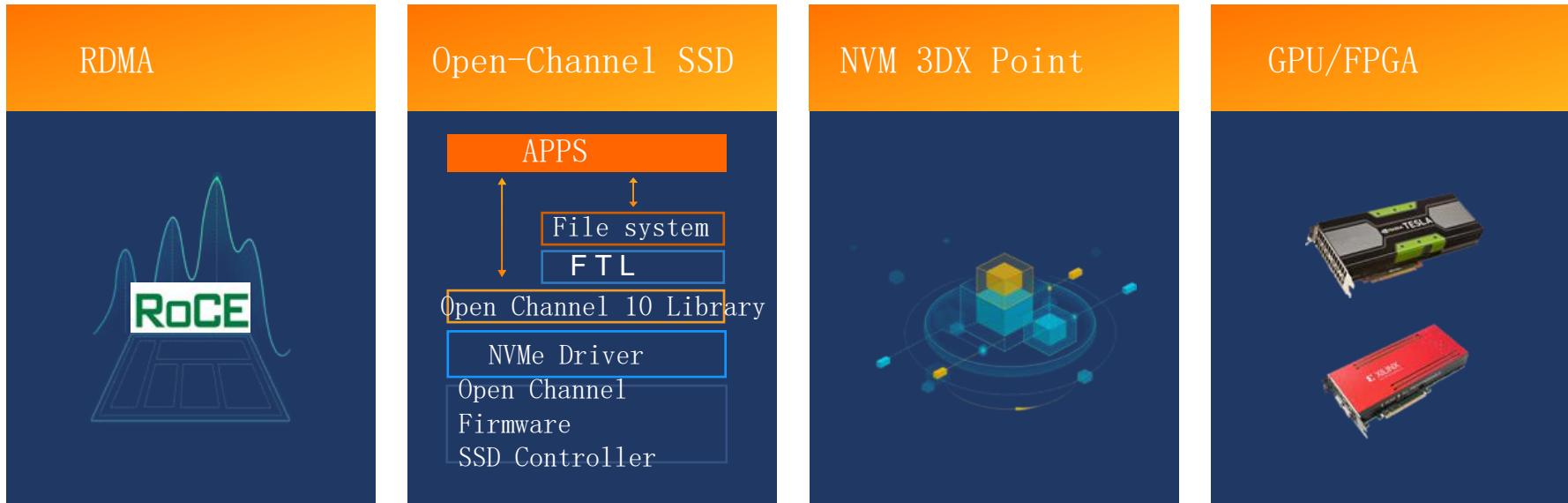


POLARDB-X (DRDS+POLARDB)

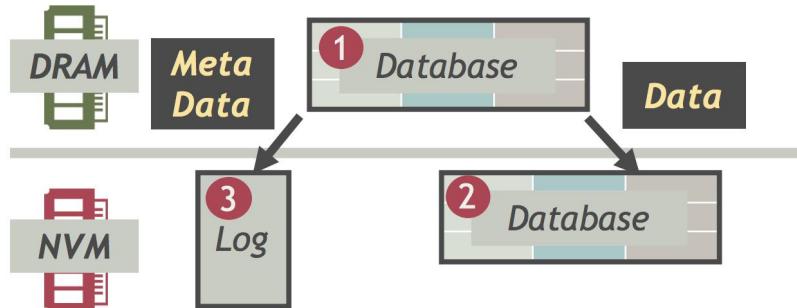
CPP-Summit

软硬件一体化(leveraging hardware)

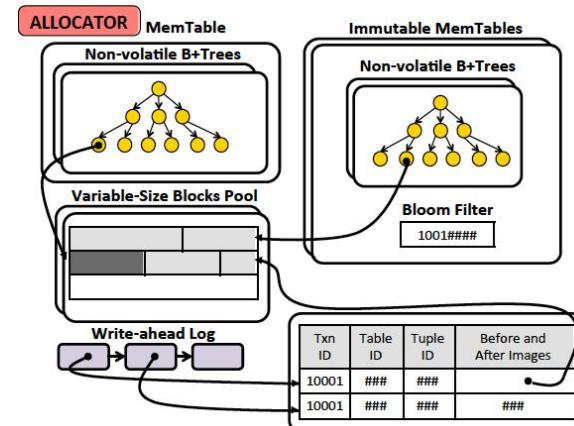
阿里云



Write behind Logging



NVM-optimized storage engine

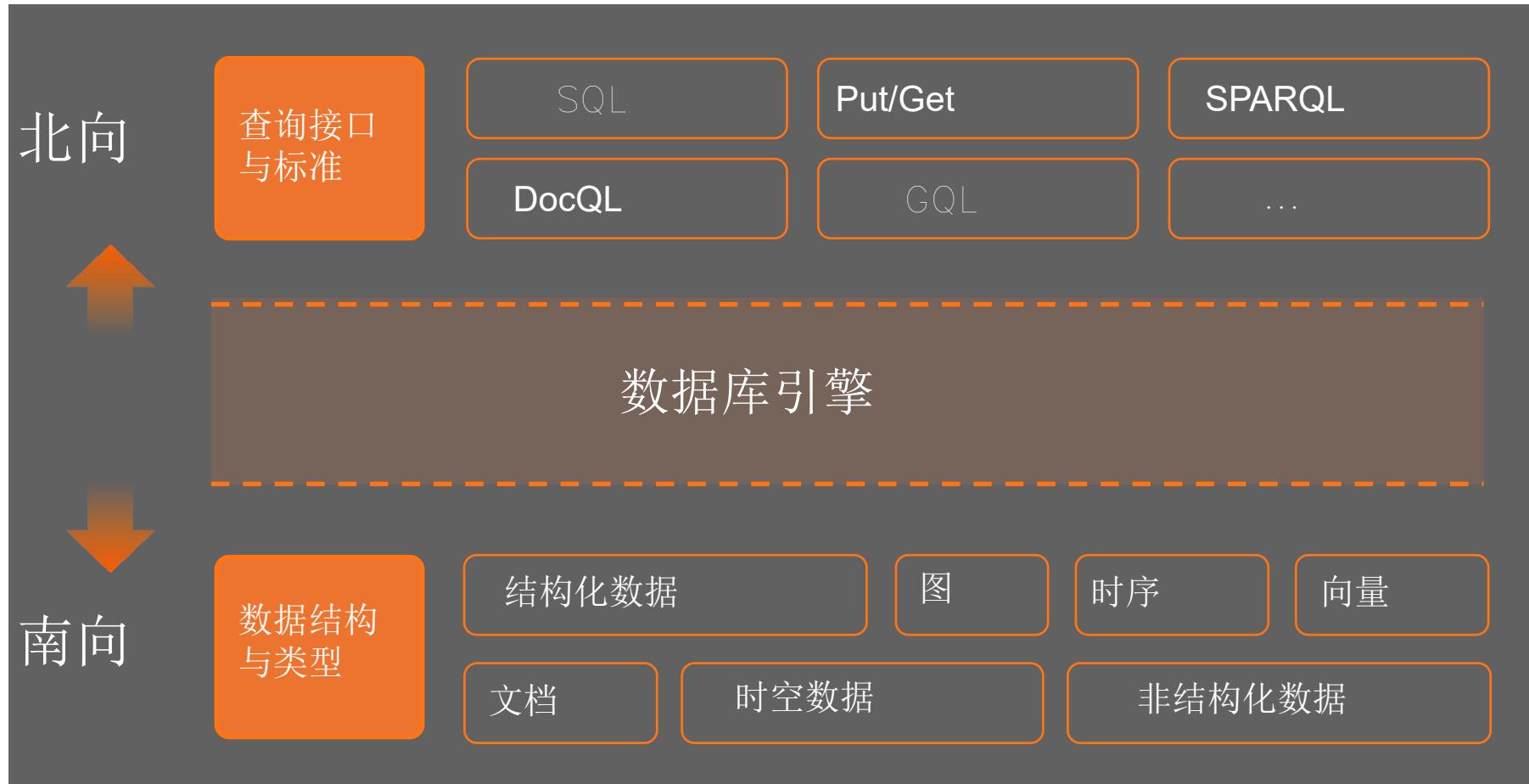


CPP-Summit

多模数据库系统(多种数据模型一体化)

阿里云

Multi-Model Database System



连少华

资深架构师



钟爱C++语言，是C++语言的资深研究者，对新技术有敏锐洞察和见解，拥有十多年一线软件架构设计和开发经验，先后在中兴通讯、深交所和金证股份任职资深开发和架构师，同时负责软件架构的设计和核心编码。目前在互联网金融企业主导公司交易系统、行情系统和量化系统的设计与开发。先后翻译了《C++代码整洁之道》和《Python代码整洁之道》。

主办方：

Boolan
高端IT咨询与教育平台

C++ Summit 2020

连少华
资深架构师

Modern C++ 整洁代码最佳实践

架构设计基础

- 架构的基本概念
- 常见架构分类
- 架构方法论

Modern C++整洁之道

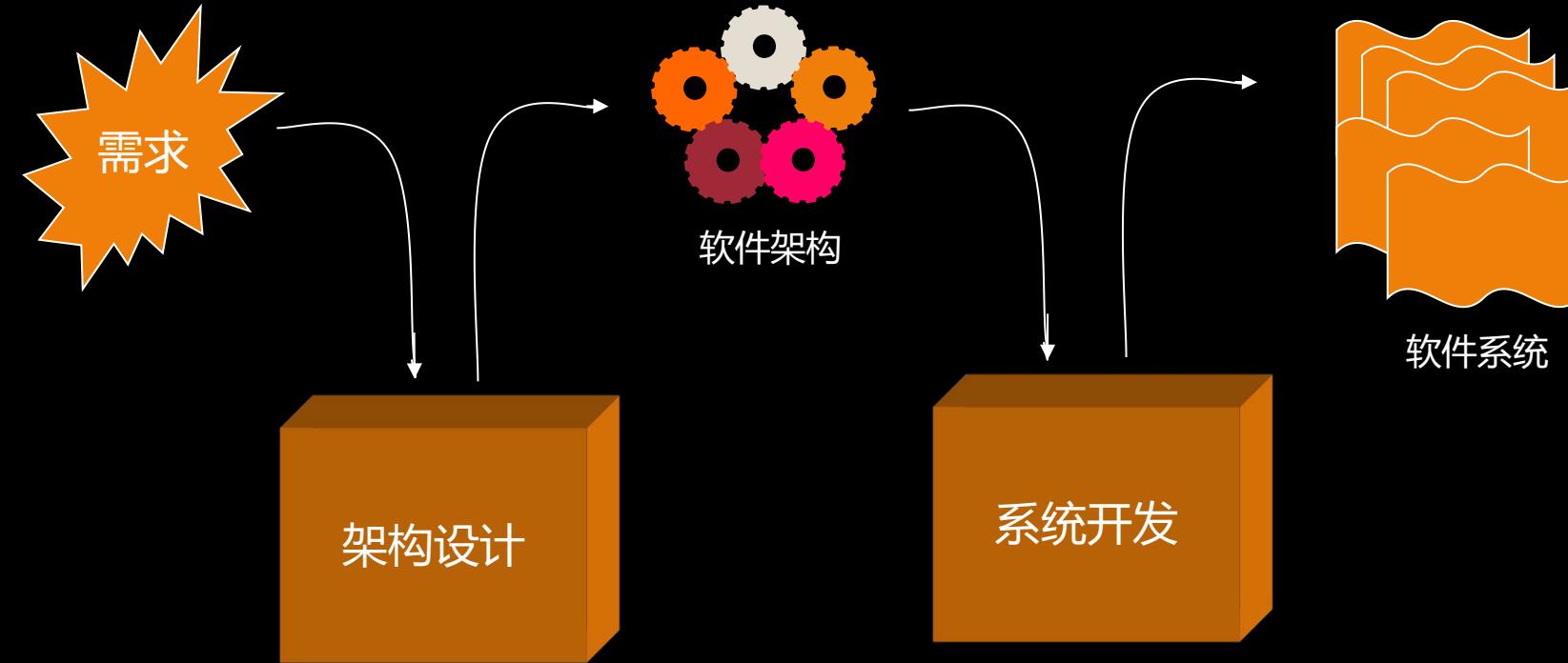
- 整洁的基本原则
- 整洁的基本规范
- Modern C++
- 面向对象的基本原则

最佳工程实践

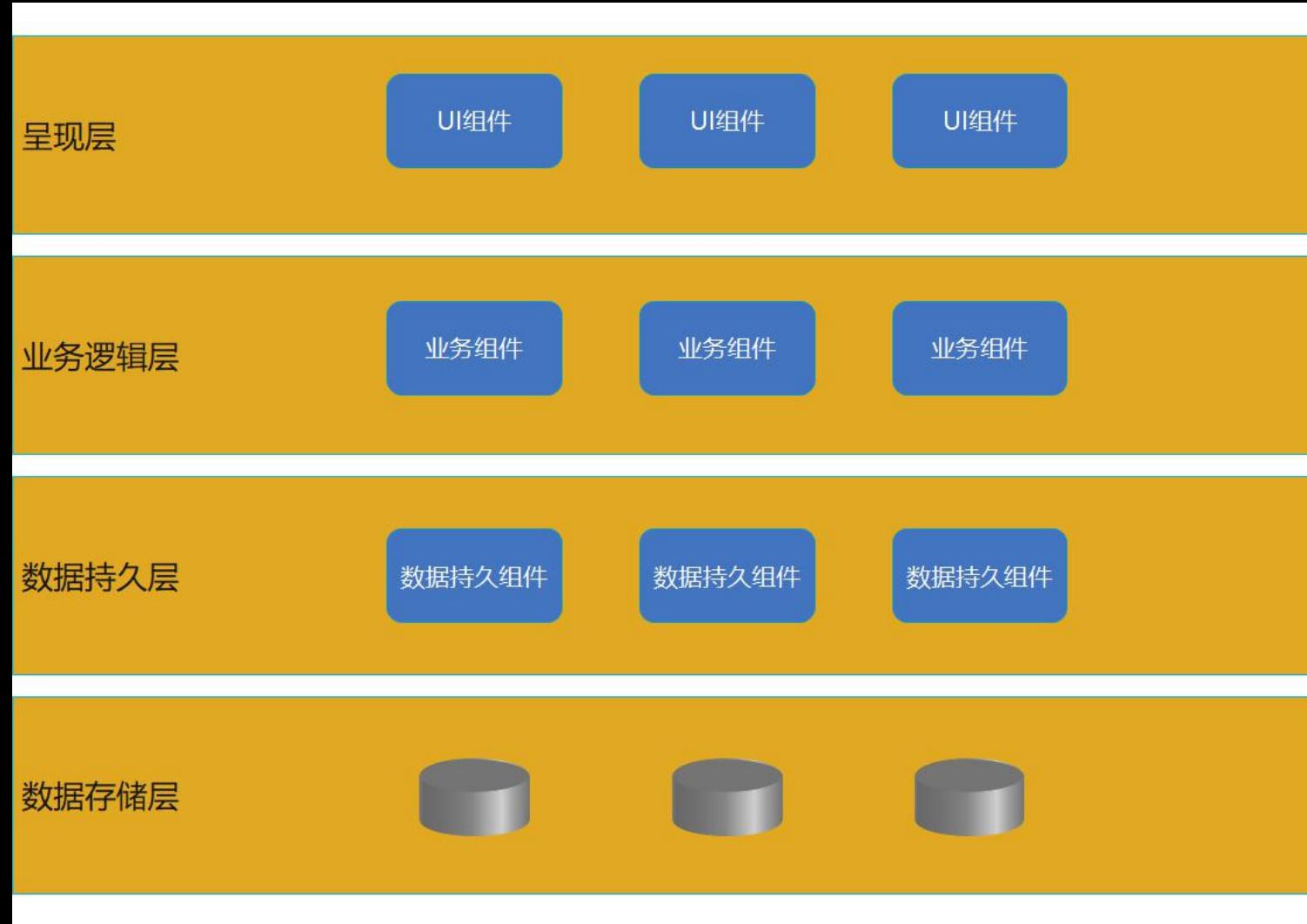
- 持续改进

- 在大街上，问一百个人可能有一百零一种说法
- 目前对软件架构的理解大致可分为剑宗和气宗两大门派



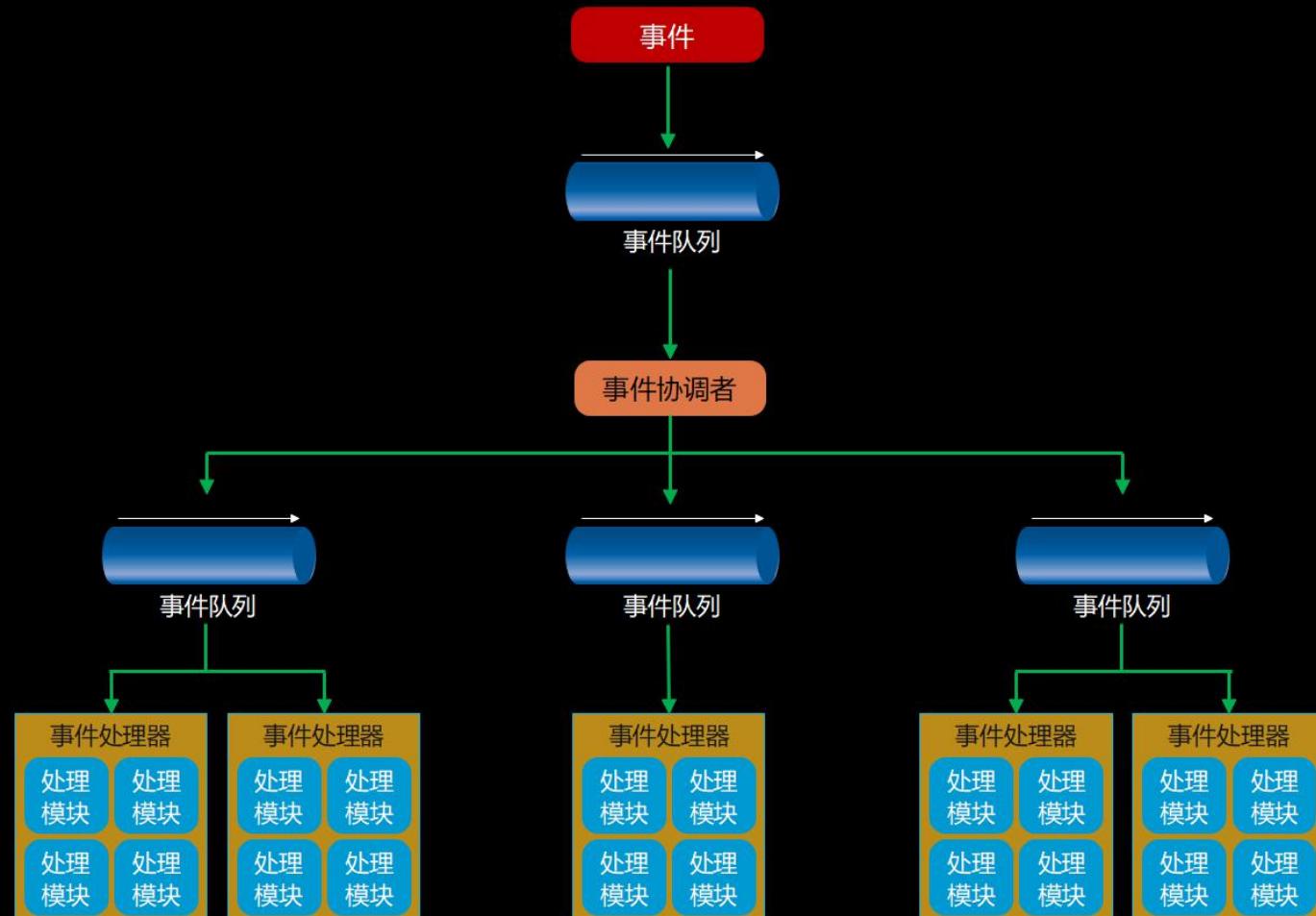


I. 分层架构



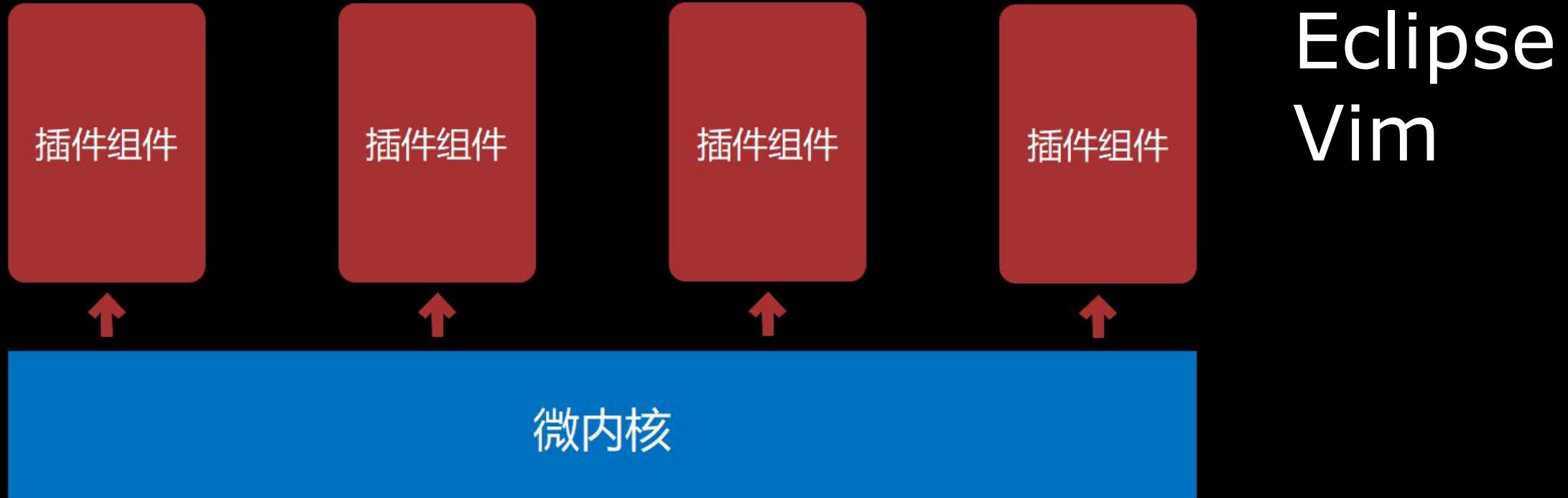
MVC
MVVM
MVP
SSH
Blockchain

II、事件驱动架构

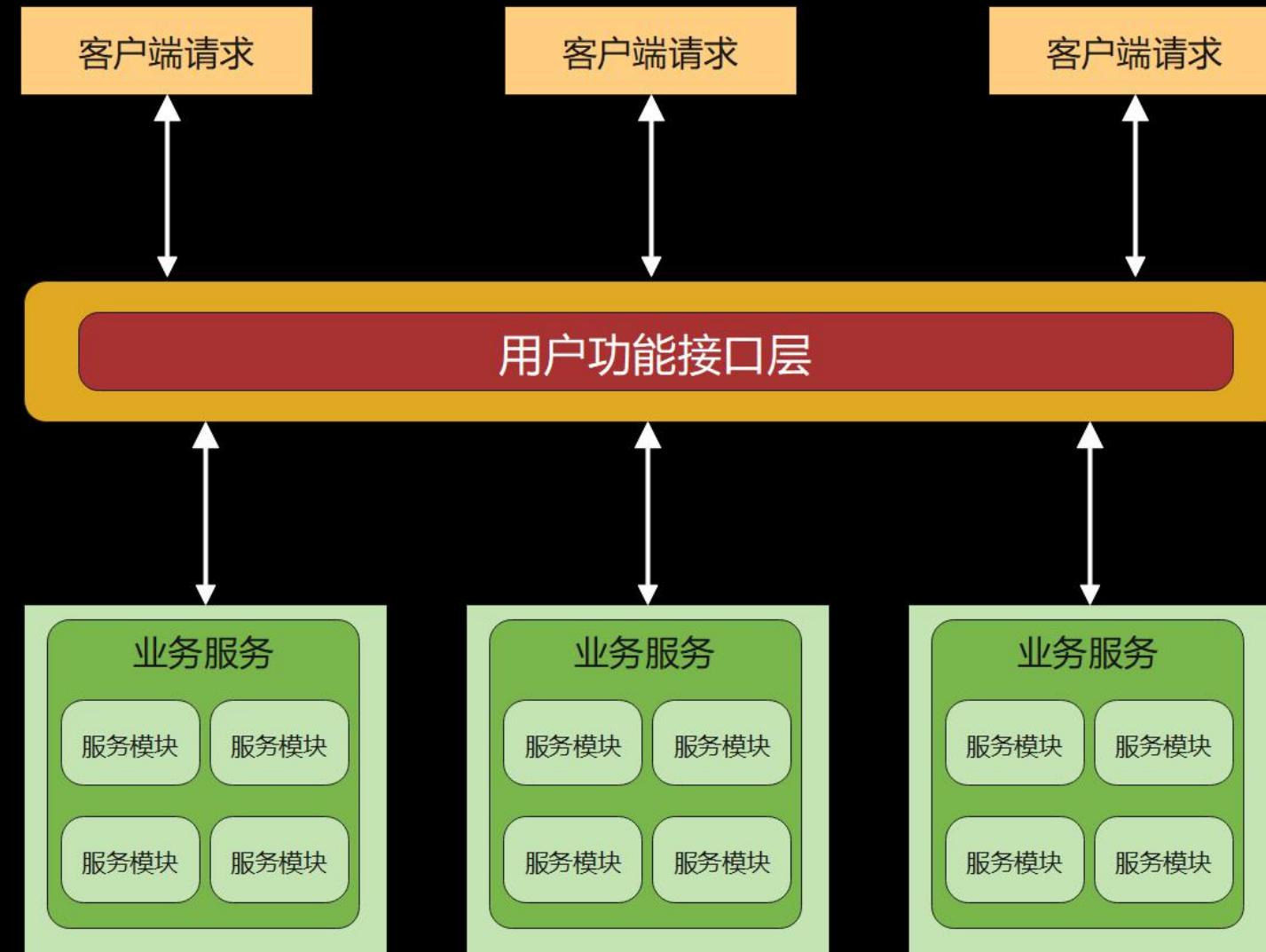


Node.js
Flink

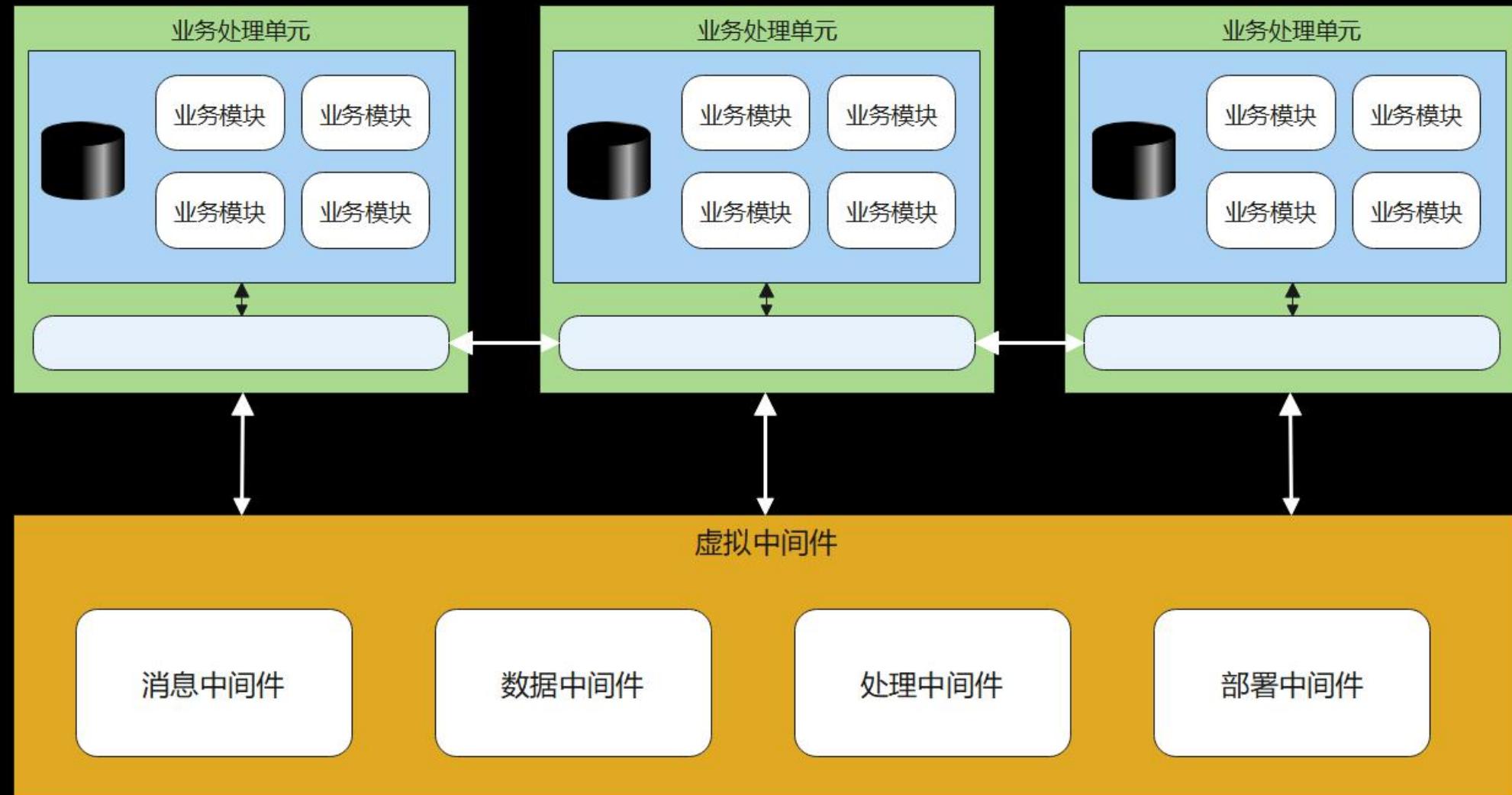
III、微内核架构

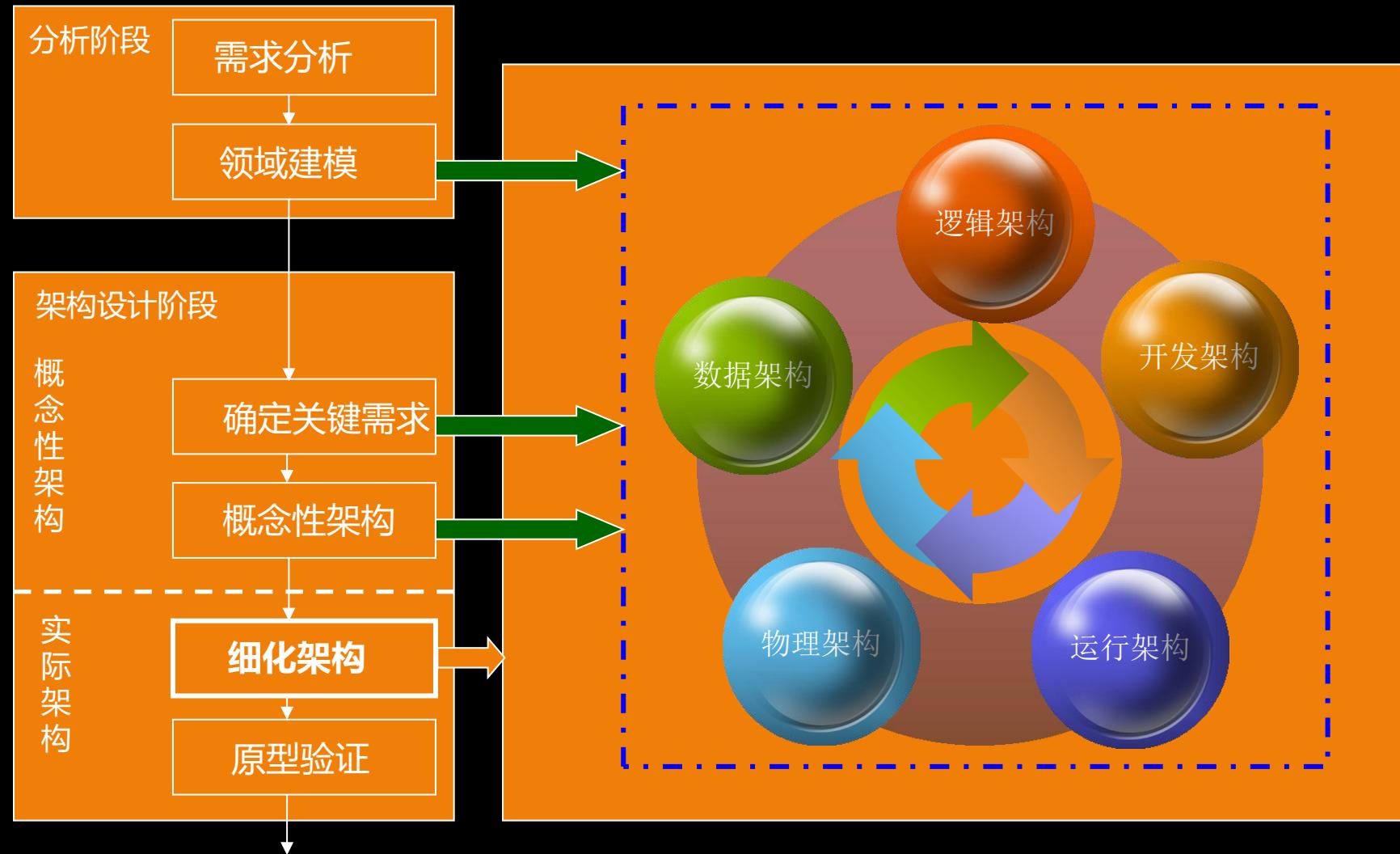


IV、微服务架构

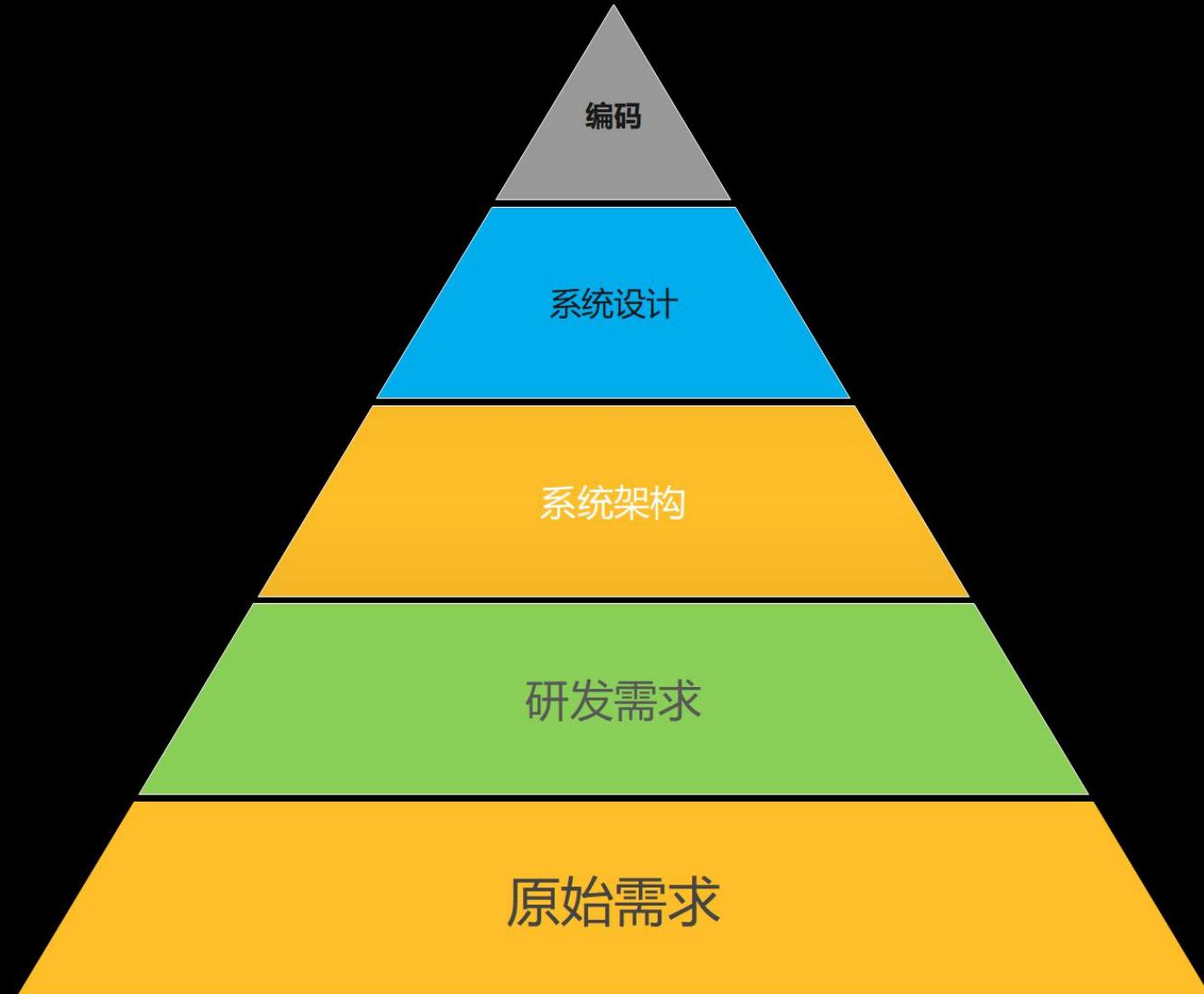


V、云架构









✓ 保持简单和直接原则 (KISS)

KISS (Keep It Simple , Stupid)原则是指代码的设计和实现越简单越好，在满足系统需求的前提下，任何没有必要的复杂都是需要避免的。因为人们喜欢简单、容易学习和易于使用的事物，同时，简单也可以缩短交付时间，降低公司成本。

大道至简。把简单的事情复杂化是没事找事情，把复杂的事情简单化才是一种能力。

对于程序员来讲，关注简单和保持简单可能是最困难的事情之一，因为程序员是最聪明的一群人，他们每天都在做着“改变世界”的事情。

✓ 不需要原则(YAGNI)

不需要 (You Are NOT Gonna Need It)指出千万不要进行过度设计,也就是说，不要写目前用不上，将来也许用得上的代码，否则，就破坏了KISS原则了。在日常讨论中，我们常常会这样“以后也许会用到这个功能...”，对于“将来也许”的功能，建议在真正有必要的时候再写代码，因为将来的事情...

✓ 避免复制原则 (DRY)

避免复制原则 (Do not Repeat Yourself)指出尽量在项目中减少重复的代码行、重复的方法和重复的模块，这就意味着项目中的每个事物都应该是唯一的。其实很多的原则、思想、模式和框架最本质的思想都是在消除重复。

任何时候都尽量避免Ctrl+C、Ctrl+V，因为重复意味着臃肿。可以想象一下，当修改一段代码时，也必须相应的修改另一段重复的代码，事实是，往往漏改或忘改那段重复的代码。

✓ 信息隐藏原则

信息隐藏原则 (Information Hiding) 指出，当一个模块（一段代码）调用另一个模块（另一段代码）时，调用者不应该知道被调用者的内部实现。这就要求被调用者把自己的实现“隐藏”起来，只提供必要的接口，所以信息隐藏是系统模块化的基本原则。

信息隐藏有很多的优点，如：限制了模块变化的范围；显著提高了模块的复用性；模块具备更好的可测试性；当修复缺陷时，对依赖模块基本没有影响。

✓ 高内聚原则

内聚是衡量内部间聚集和关联的程度，高的意思是内部间的关系要简单明了，不要牵强附会。高内聚是信息隐藏原则的扩展，是从功能角度进行度量的。内聚可分为偶然内聚、逻辑内聚、时间内聚、过程内聚、通信内聚、顺序内聚、功能内聚，功能内聚是最强的内聚。

✓ 松耦合原则

松耦合是衡量模块间相互联系的紧密程度。耦合的强弱与模块间接口的复杂性、调用方式和传输数据有直接关系。耦合可分为非直接耦合、数据耦合、标记耦合、控制耦合、外部耦合、公共耦合、内容耦合。

✓ 小心优化原则

小心优化原则建议没有明确的性能要求，就避免优化。因为不成熟的优化是编程中绝大部分问题的根源。有些开发人员，在没有找到问题所在甚至没有完成编码前，就进行各种各样的优化，其实很浪费时间，也无法解决根本性的问题。

✓ 名称应该自解释

应望文知意，长度适中，使用简单，能自我解释和描述。如：flag、list、data、vec、num、Info等都是不太好的名字，像isRunning、orderInfo、orderList、productNumber等是比较好的命名。

示例：

```
unsigned int num;  
bool flag;  
std::vector<Book> bookVector;  
std::string info;  
unsigned int totalPriceOfCustomerBooksToday;
```



```
unsigned int numberofBook;  
bool isRunning;  
std::vector<Book> books;  
std::string orderInfo;  
unsigned int priceofBooks;
```

✓ 使用域中的名称

如果有领域驱动设计 (Domain-Driven Design, DDD)环节，那么在写代码之前，建议先仔细阅读DDD中的命名，DDD中的术语、名称比较专业，同时命名也保证了统一。

✓ 避免冗余的名称

类（或模块）中的数据成员或函数成员名称尽量不要带有类（或模块）的名称，也不要包含类型的名称。

示例：

```
enum class PersonSex:int
{
    Male=0,
    Female=1
};

class Person:public std::enable_shared_from_this<Person>
{
public:
    string getPersonName() const;
    int setPersonName(string name);

    int getPersonAge() const;
    int setPersonAge(int age) const;

    PersonSex getPersonSex() const;
    int setPersonSex(PersonSex newSex) const;

private:
    string personName;
    int personAge;
    PersonSex personSex;
};
```

VS

```
enum class Sex:int
{
    Male=0,
    Female=1
};

class Person:public std::enable_shared_from_this<Person>
{
public:
    string getName() const;
    int setName(string newName);

    int getAge() const;
    int setAge(int newAge);

    Sex getSex() const;
    int setSex(Sex newSex);

private:
    string name;
    int age;
    Sex sex;
};
```

✓ 避免晦涩难懂的缩写

在单词不是很长的时候尽量不要使用自定义缩写，一是有歧义，二是记忆和学习的成本太高了，如：idx、bmw、rfg等；但是可以使用一些与行业相关的“知名缩写”，如：MD、TRD、TGW、UDP、TCP等

示例：

```
std::size_t idx;  
Car ctw;  
unsigned int nBottles;  
bool rfg;
```

VS

```
std::size_t index;  
Car carToWash;  
unsigned int bottleAmount;  
bool runFlag;  
  
unsigned int trdAmount;  
unsigned int tgwStatus;
```

✓ 避免相同的名称用于不同的目的

就像一个公司中有两个同名的人一样，看到名字时不知道这个名字到底指的是谁。

✓ 避免匈牙利命名和命名前缀

尽量不要把类型信息加入到变量的名称中，因为经常会出现变量类型与名称表示的类型不一致的情况——类型前缀不可信，如：pszName、strInfo、iSize、fFlag等。现在很多高级的IDE都支持了智能提示，所以尽早的抛弃匈牙利命名法吧。

示例：

```
bool fEnable;
unsigned long long ullContainerSize;
int iIndex;
char * pszTitle;
vector<Book> vecBook;
double dLastPrice;
float fPrice;
char * gsc_pszPrefixString;
```

VS

```
bool isEnabled;
unsigned long long containerSize;
int index;
char * title;
vector<Book> books;
double lastPrice;
float price;
char * prefixString;
```

✓ 代码应该自注释

请理解一句话——真相只能在代码中找到。所以，能不写注释就不要写注释，因为事实已经证明了注释往往与代码不一致，这个时候，我们应该相信代码还是应该相信注释呢？

✓ 不要为易懂的代码写注释

```
/*
 * @brief proxy_manager
 * It manages xpub_xsub_proxy and router_proxy instances
 * It can not be inherited
 */
class proxy_manager final
{
private:
    //Define proxy_vector type
    using proxy_vector=std::vector<proxyIPtr>;
public:
    //...
    //Initialize proxy_manager instance
    error_code Initialize() noexcept;
    //Start to run proxy_manager instance
    error_code Run() noexcept;
    //Release all resources allocated in Initialize function
    error_code Uninitialize() noexcept;

private:
    //Initialize flag
    std::atomic_bool isInitialized=false;
    //Running flag
    std::atomic_bool isRunning=false;
    //The Vector with proxy instance
    proxy_vector proxies;
};
```

VS

```
class proxy_manager final
{
private:
    using proxy_vector=std::vector<proxyIPtr>;
public:
    //...

    error_code Initialize() noexcept;
    error_code Run() noexcept;
    error_code Uninitialize() noexcept;

private:
    std::atomic_bool isInitialized=false;
    std::atomic_bool isRunning=false;
    proxy_vector proxies;
};
```

✓ 不要通过注释禁用代码

被禁用的代码增加了代码的混乱程度，却没有带来任何的好处，提交到版本管理系统的代码“永久”不会丢失。

✓ 不要写块注释

文件头的创建、更改记录、版权声明、版本控制、大段大段的//或/*...*/的注释。

```
/* Copyright (C)
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 */
//#####
//Change log:
//2006-06-14 (dc cheng) fix bug #11086
//2006-06-14 (nn xu) fix bug #11072
//2006-05-01 (y yu) create file
//#####

/***
 * @file memory_pool.cpp
 * @brief memory pool
 * @author shlian
 * @version 1.0
 * @date 2020-09-28
 */

```

```
/**
 * @brief MQ proxy
 */
class mq_proxy final
{
public:
    //-----
    //public interface
    //-----

    //.....
protected:
    //-----
    //override or protected methods
    //-----

    //.....
private:
    //-----
    //private member functions
    //-----

    //.....
    //-----
    //private data members
    //-----

    //.....
}
```

✓ 有些注释是必要的

从源代码生成帮助文档时，可以适当的写一些简要的注释，如经常使用的Doxygen工具提供的几种注释风格。

```
/**  
 * @brief proxy interface  
 */  
class mq_proxy  
{  
public:  
    //...  
    /**  
     * @brief Initialize the proxy,allocate resources needed  
     *  
     * @param thread_number the number of thread to create  
     *  
     * @return ok:successful,other:error  
     */  
    error_code Initialize(int thread_number) noexcept;  
  
    /**  
     * @brief Start the proxy  
     *  
     * @return ok:successfull,other:error  
     */  
    error_code Start() noexcept;  
  
    /**  
     * @brief Stop the proxy  
     *  
     * @return ok:successfull,other:error  
     */  
    error_code Stop() noexcept;  
  
    /**  
     * @brief Wait the proxy to complete all jobs  
     *  
     * @return ok:successful,other:error  
     */  
    error_code Wait() noexcept;  
  
    /**  
     * @brief Uninitialize the proxy to releace resources  
     *  
     * @return ok:successfull,other:error  
     */  
    error_code Uninitialize() noexcept;  
};
```

✓ 只做一件事情

一个函数应该做一件事情，且应该仅仅做好这一件事情。如果函数体量比较大、没有合适的名字、人为了划分了几个段落、圈复杂度比较度、入参比较多等，都是函数做了太多事情的标志。

✓ 让函数尽可能的小

函数体应该尽可能的小（参见第一条）。函数调用开销并不是系统的瓶颈，并且现在的编译器会优化掉函数的调用开销，编译器比你聪明得多。

✓ 使用容易理解的名称

函数名称应该明确的表达清楚函数的目的，而不是过程。

✓ 函数的参数和返回值

参数应尽可能的少，应该避免参数间存在依赖关系，尽量不使用标志性的参数。

- ✓ 尽量使用C++的std::string、std::stream替代C风格的char*

不安全，且性能上的“提升”不足以掩盖其带来的问题。

- ✓ 避免使用printf、str、mem系列的函数，如：snprintf、strncpy、memcpy等

不安全，且性能上的“提升”不足以掩盖其带来的问题。

```
const char *prefix="CPP-Summit 2020";
const unsigned int title_length=50+1;
char *title=new char[title_length];

strcpy(title,prefix);
//strncpy(title,prefix,title_length);
//memcpy(title,prefix,title_length);
//memcpy(title,prefix,strlen(prefix)+1);

snprintf(title,title_length,"%s%s",prefix,"shenzhen");
printf("%d\n",title);
```

VS

```
const char *prefix="CPP-Summit 2020";
std::ostringstream oss;
oss<<prefix<<"shenzhen";
string title(oss.str());
cout<<title<<endl;
```

✓ 使用标准库的容器替代C风格的数组

std::array比较安全，不会越界，且兼容STL接口，并且还与C风格的数组兼容。

✓ 使用C++类型转换代替C风格的强制转换

尽量避免类型转换！！C++类型转换会在编译期进行检查，而C风格的则不会。

✓ 尽量避免使用宏

尽量避免使用宏，宏定义的函数并不能带来效率上的提升，并且会导致文件体量庞大。

```
std::array<int,20> topNumbers;
topNumbers[0]=100;
topNumbers[19]=-100;
topNumbers[200]=30;

for_each(topNumbers.begin(),topNumbers.end(),[](int element){
    cout<<element<<endl;
});

int *number=topNumbers.data();
for(size_t i=0;i<topNumbers.max_size();++i)
{
    cout<<number[i]<<endl;
}
```

```
const char *title="CPP-Summit 2020";

int *value1=(int*)title;
char *value2=(char *)title;
float *value3=(float*)title;
void *value4=(void*)title;

int *val1=static_cast<int*>(title);
char *val2=static_cast<char*>(title);
char *val3=const_cast<char*>(title);
void *val4=reinterpret_cast<void*>(title);
```

- 资源申请即初始化(RAII)，即“构造时获得，析构时释放”
- 使用指针时，请优先使用智能指针，如：std::shared_ptr、std::weak_ptr、std::unique_ptr

- 尽量避免手动管理内存，使用堆栈内存、make_xxx函数、使用stl容器、池

```
class summit final
{
public:
    summit()
    {
    }

    ~summit();
    summit(const summit &other);
    summit & operator=(const summit &other);

    summit(summit && other);
    summit & operator=(summit &&other);

private:
    string title="CPP-Summit 2020";
    unsigned int beginDate=20201204;
    unsigned int endDate=20201205;
    string address="shenzhen";
    vector<string> titles;
    unsigned int numberOfConferee=2000;
};

void build_summit()
{
    auto summitInstance=std::make_shared<summit>();

    auto summitInstance2=std::make_unique<summit>();
}
```

→ “零原则”
VS

```
class summit final
{
public:
    summit()
    {
    }

private:
    string title="CPP-Summit 2020";
    unsigned int beginDate=20201204;
    unsigned int endDate=20201205;
    string address="shenzhen";
    vector<string> titles;
    unsigned int numberOfConferee=2000;
};

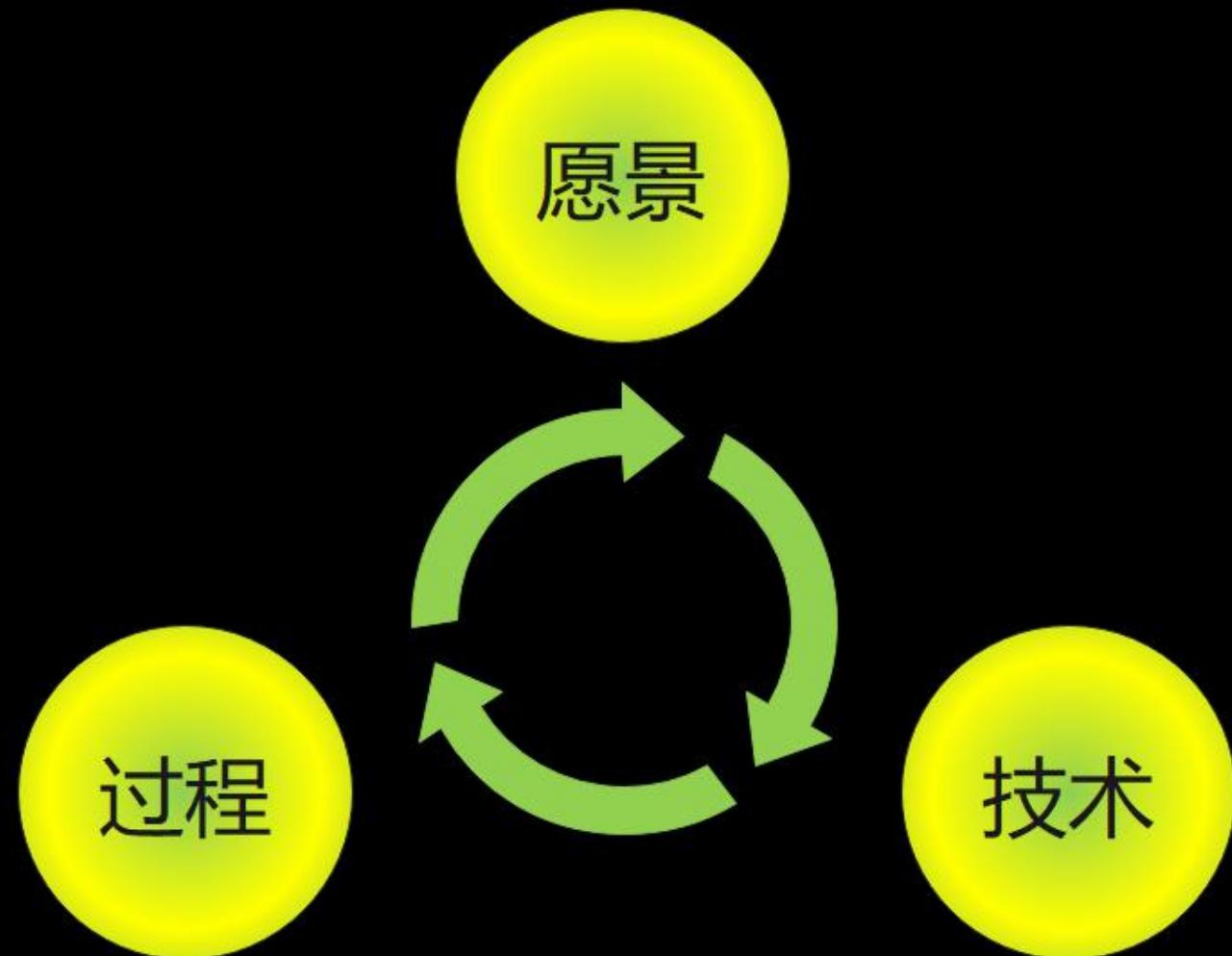
void build_summit()
{
    auto summitInstance=std::make_shared<summit>();

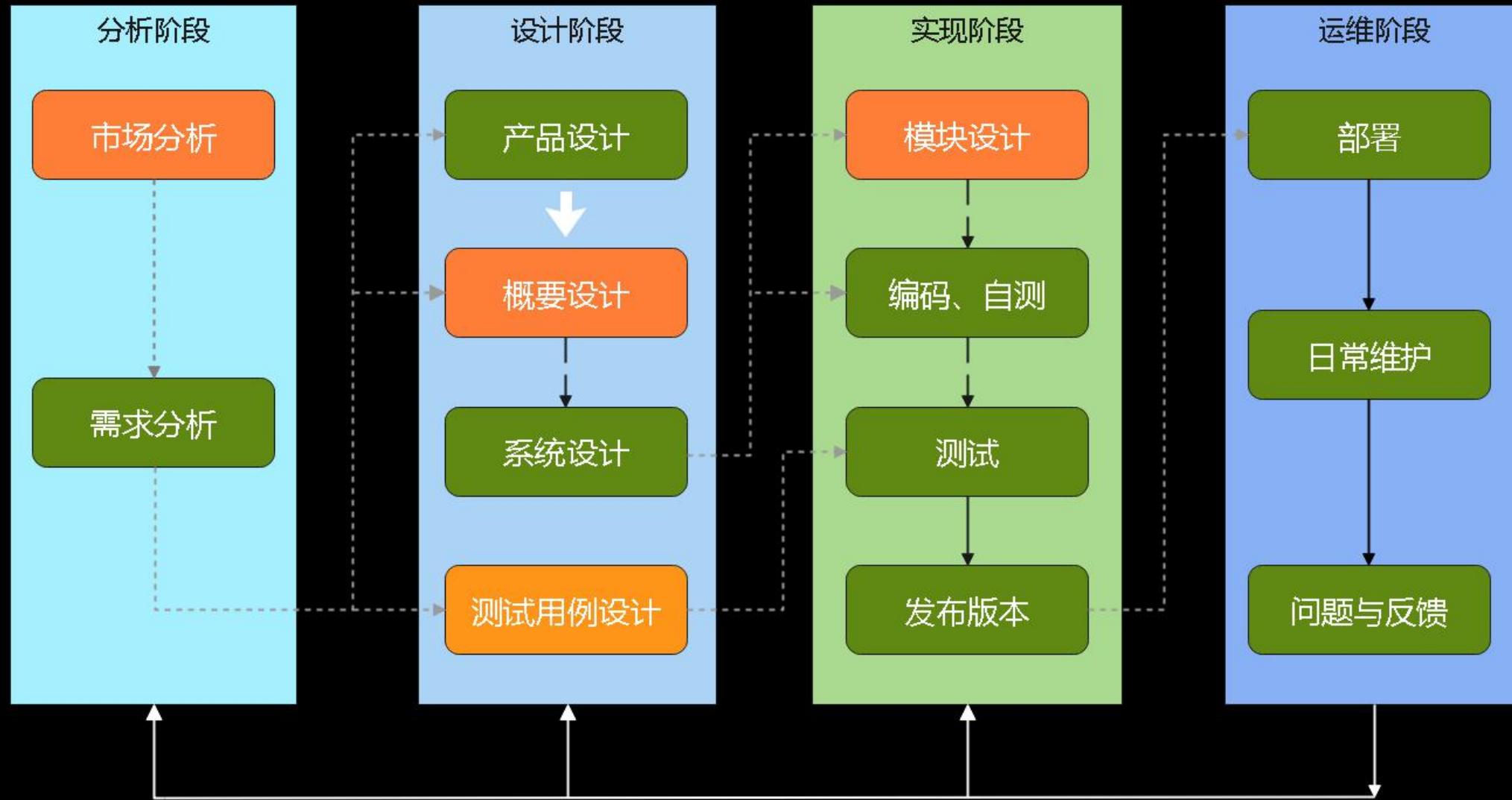
    auto summitInstance2=std::make_unique<summit>();
}
```

- 必要的时候，使用自动类型推导
- 尽可能的使用编译时计算，如：constexpr、template
- 熟悉stl库及相关算法，尽量避免使用for和while循环
- 尽量使用lambda而不是functor
- 使用Type-Rich
- 学习、使用并倡导Modern C++

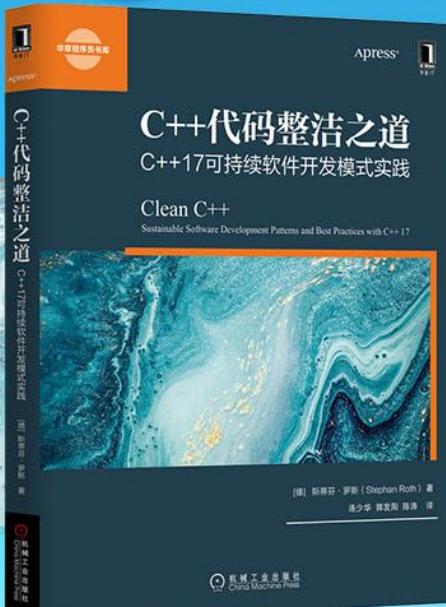
```
struct Money{  
    long double value;  
};  
  
constexpr Money operator"" _CNY(long double value)  
{  
    return Money{value=value};  
}  
  
int main(int argc, char *argv[])  
{  
    auto moneys={1.0_CNY,10.0_CNY,125.0_CNY,300.0_CNY};  
    long double sum=0;  
  
    for_each(moneys.begin(),moneys.end(),[&sum](const auto & money){  
        sum+=money.value;  
        cout<<money.value<<endl;  
    });  
    cout<<"sum="<<sum<<endl;  
}
```

- 让类尽可能的小，尽量避免出现“工具类”、“万能类”
- 单一职责 (Single Responsibility Principle , SRP)
- 开闭原则 (Open-Close Principle , OCP)
- 里氏替换原则 (Liskov Substitution Principle , LSP)
- 接口隔离原则 (Interface Segregation Principle , ISP)
- 无环依赖原则 (Acyclic Dependencies Principle , ADP)
- 依赖倒置原则 (Dependence Inversion Principle , DIP)
- 迪米特法则，最少知识原则 (Least Knowledge Principle , LKP)
- 避免贫血类
- 优先使用组合而不是继承
- 尽量避免使用静态成员，单例可能并不是一个好的设计模式。



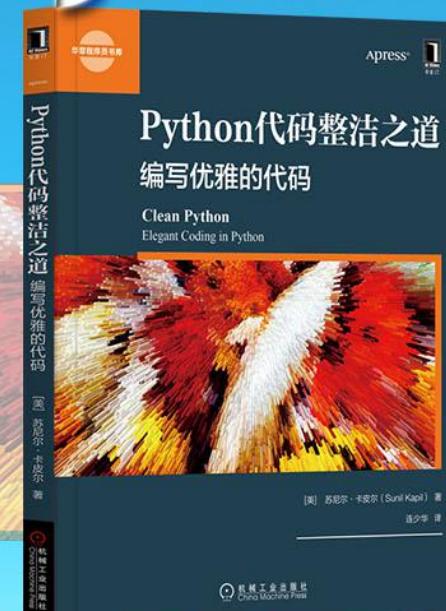


C++



代码整洁之道

Python



掌握高效的现代C++编程法则；
学会应用C++设计模式和习惯用法；
创建可维护、可扩展的软件

通过示例介绍如何编写更加整洁、
优雅的Python代码，并介绍一些非
常有用的工具

Thank You!

刘光聪

资深技术咨询师



资深技术咨询师，全栈工程师。擅长大系统软件的领域建模、架构演进与重建、遗留系统重构与开发者测试。拥有10多年大型电信软件系统的工作和咨询经验，包括大型嵌入式软件平台、无线4G/5G业务软件、机器学习计算平台等相关领域。

主办方：

Boolan
高端IT咨询与教育平台

C++ Summit 2020

刘光聪

资深技术咨询师

面向领域模型的C++实现模式

主要内容

1. 领域模型落地面临的技术挑战
2. C++实现领域模型的实现模式
3. 总结



领域模型落地面临的技术挑战

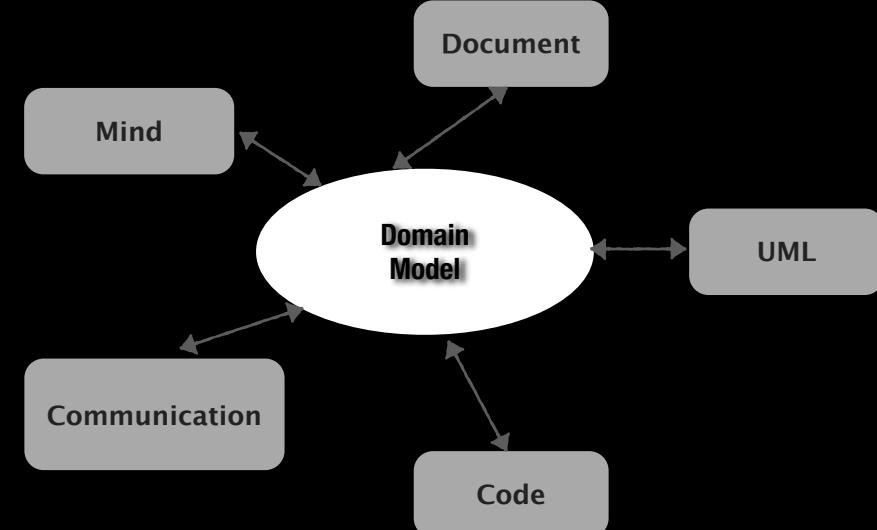
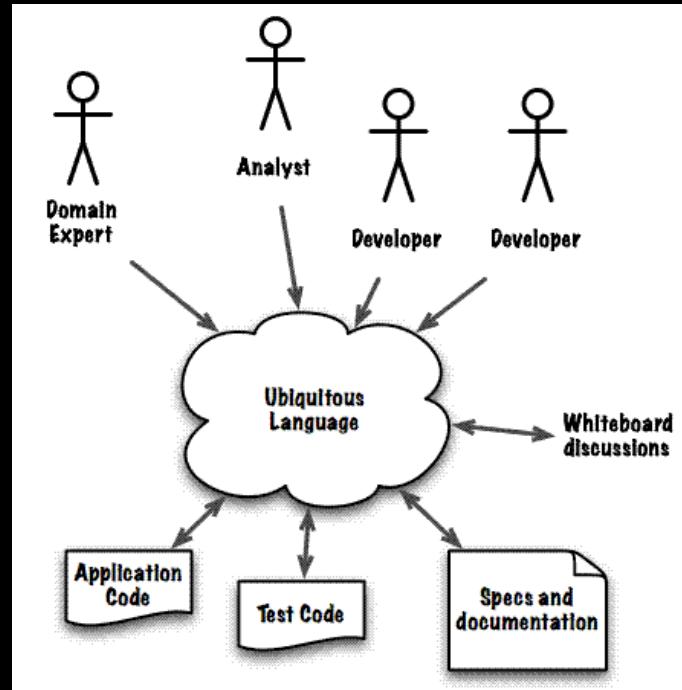
领域驱动设计概述



领域驱动设计（Domain Driven Design: DDD）：
用演进式的领域模型将软件设计与实现连接起来，用来满足复杂领域的软件开发方法。

- 将软件开发项目聚焦在核心领域模型上；
- 通过领域模型将业务专家和技术专家协同起来；
- 用迭代式演进的领域模型来应对设计和开发中的复杂性；

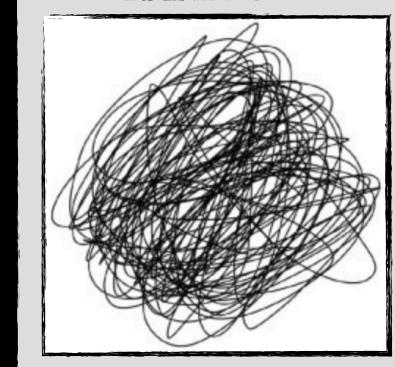
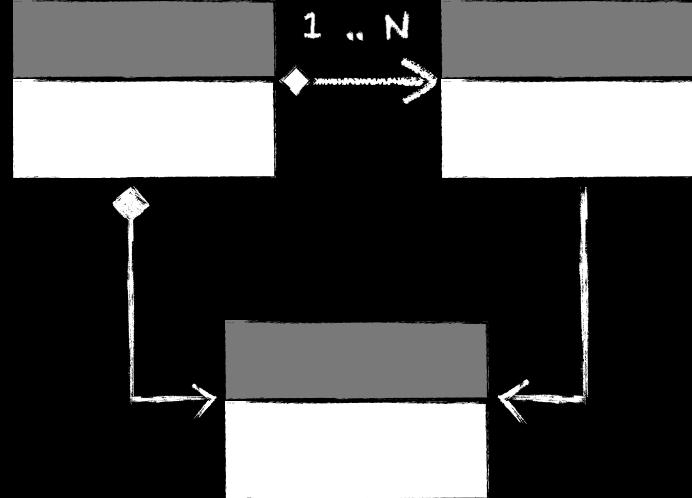
领域建模



领域模型：

- 对领域与实现的相关知识进行严格的组织和选择性抽象，识别业务主要变化方向，凸显业务的**设计本质**；
- 在软件开发过程中，是团队进行沟通的**通用语言**，设计人员和实现人员能够共同理解；
- 设计文档和代码与领域模型保持**高度对齐，同步演进**。

领域模型落地的困境



代码从逻辑到物理上难以和模型对应

- 如果一个领域模型，不能映射到代码上；或者模型和代码之间的映射如果过于复杂，晦涩，难以理解。那么这个模型是没有价值的，软件的正确性也是值得怀疑的；
- 如果编写代码的人认为他们不对模型负责；或并不理解如何让模型在一个程序中发挥作用；那么模型对于软件来说根本上就没有用；
- 如果开发人员没有意识到改变代码会同时改变模型，那么他们的重构工作只会减弱模型的作用，而不是增强；

实现领域驱动设计



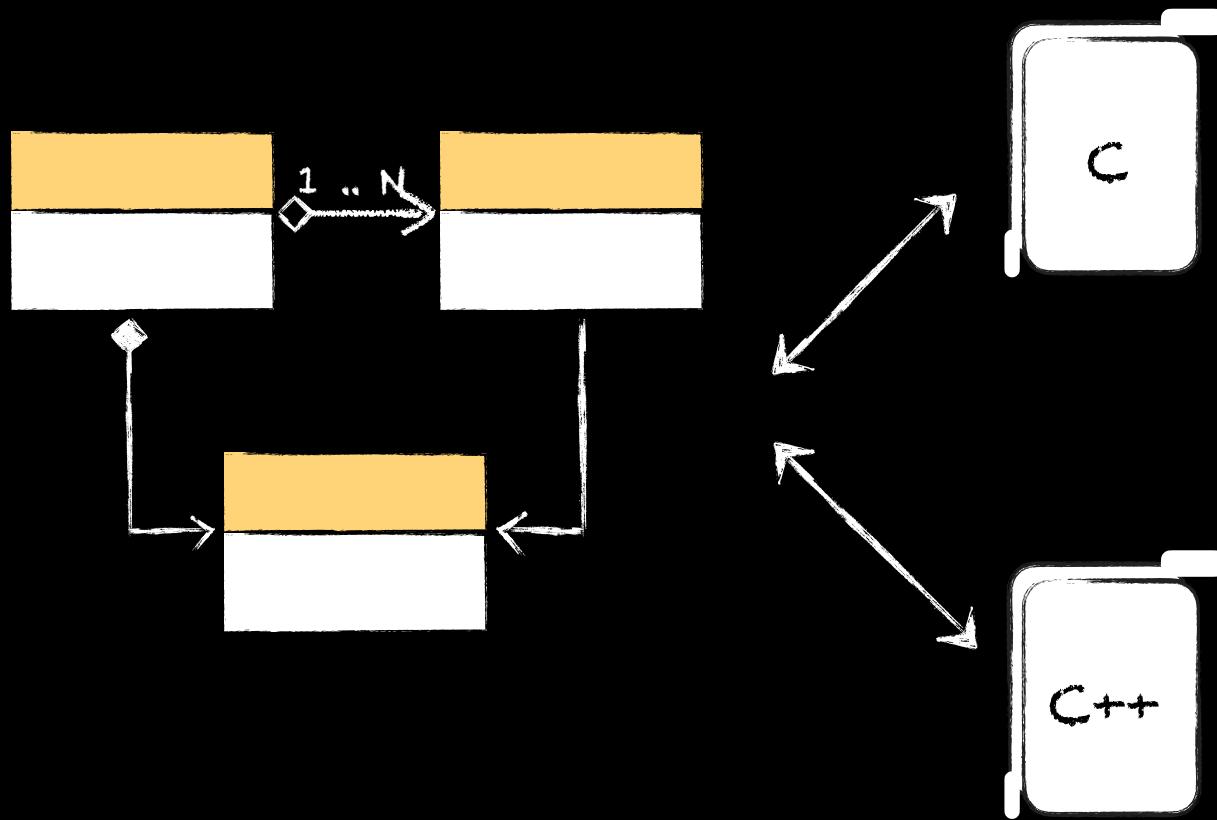
实现领域模型：

- 重新阐释和发展“领域驱动设计”；
- 更易于理解的案例；面向JAVA语言的示意性代码；
- 如何让代码对模型进行有效表达，指导的不够深入具体；

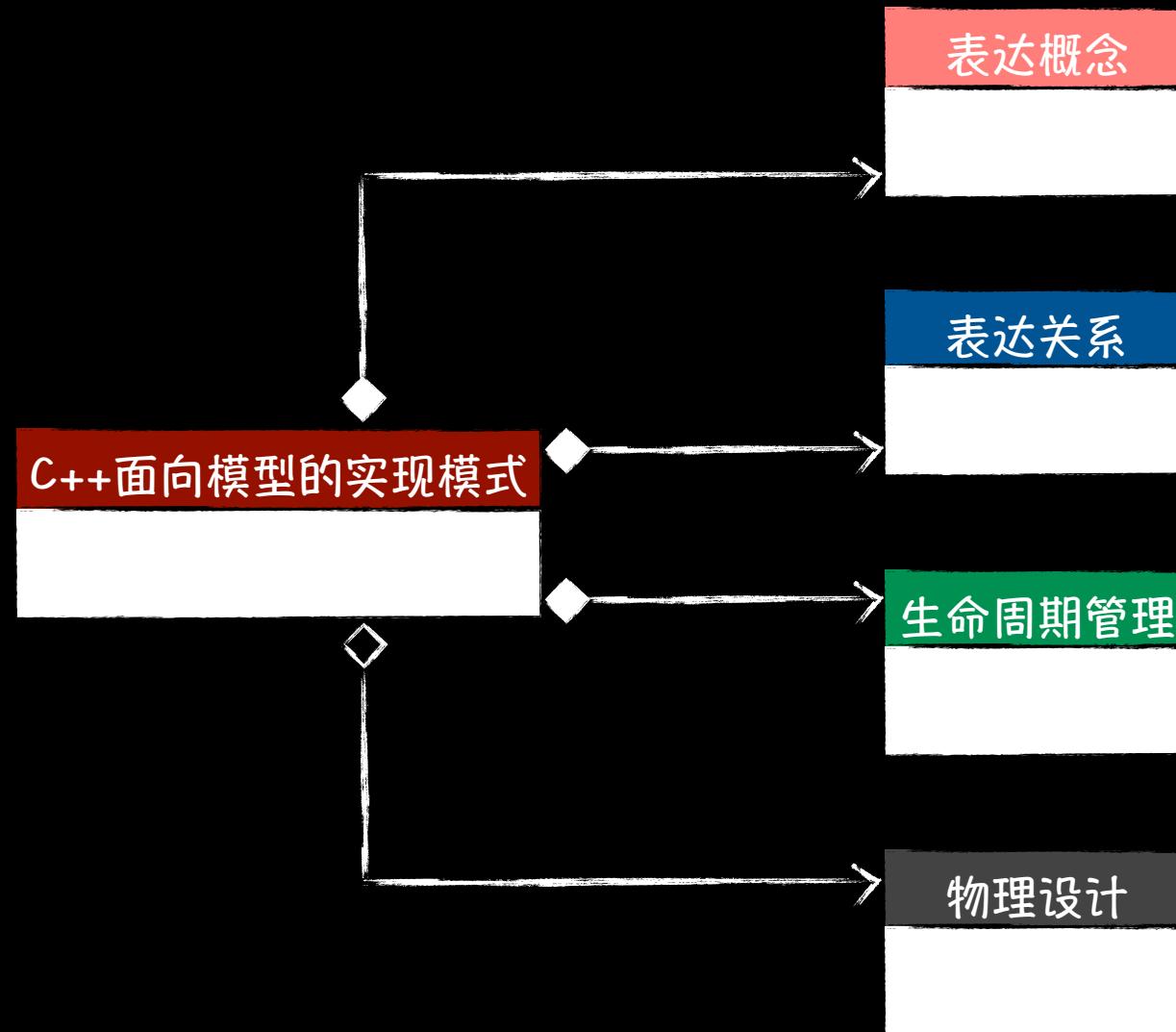
实现模式：

- 实现模式介于设计模式和语言手册之间；设计模式更多关注的是对象之间如何组织、关联的决策，是高于具体编程语言的设计决策；
- 面向JAVA语言，但没有按照“实现模型”的角度进行组织；

C++ 语言面向模型的实现模式



- 面向C++语言层面的实现模式；
- 按照“实现模型”的角度进行组织；
- 从模型到代码双向映射的最佳实践；
- 不关注模型为何而来，只关注模型如何实现；





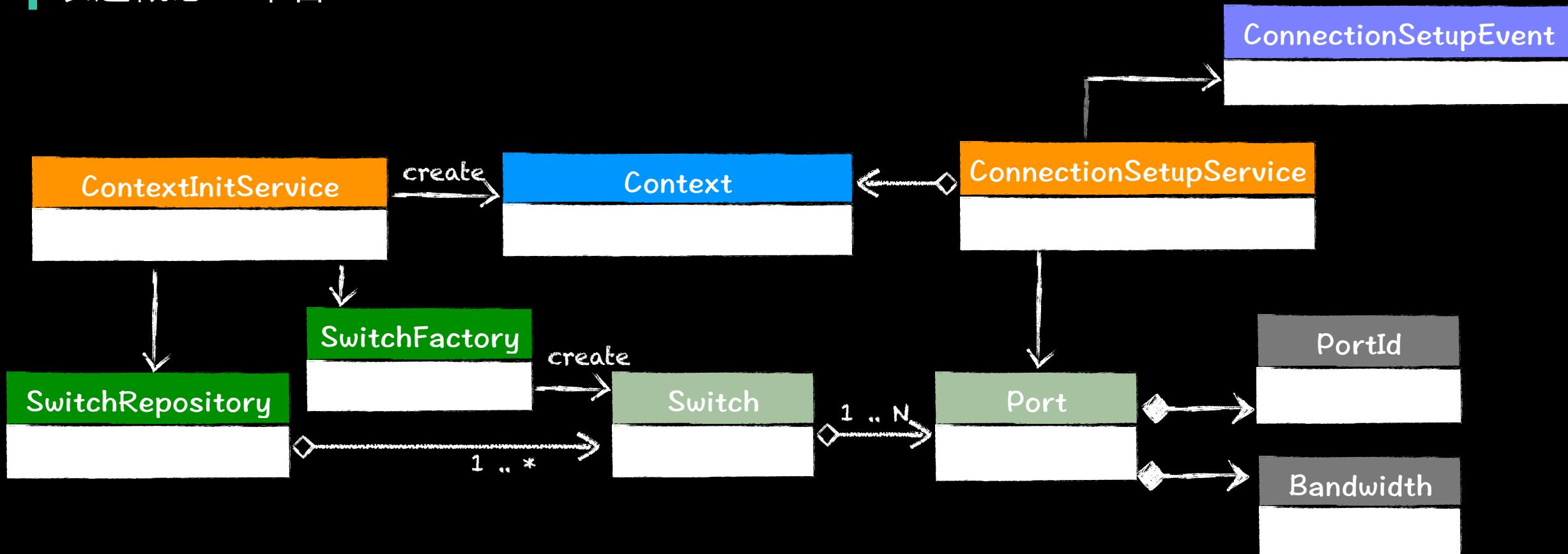
C++实现领域模型的实现模式：表达概念

表达概念

Port	
-	id : PortId
-	bandwidth : BandWidth
+	Port(id : PortId, s : Switch)
+	connect(peer : PeerNode) : Connection
+	release(conn : Connection)

- **命名**: 兼顾领域语义和模型角色;
- **实现选型**: 类 优先于 POD 优先于 基本类型;
- **构造与析构**: 服务于生命周期管理, 声明依存关系;
- **行为**: 对外接口 先于 对内复用;
- **属性**: 区分 “我是谁”, 体现依赖关系;
- **物理结构**: 独立文件, 命名一致, 格式统一;

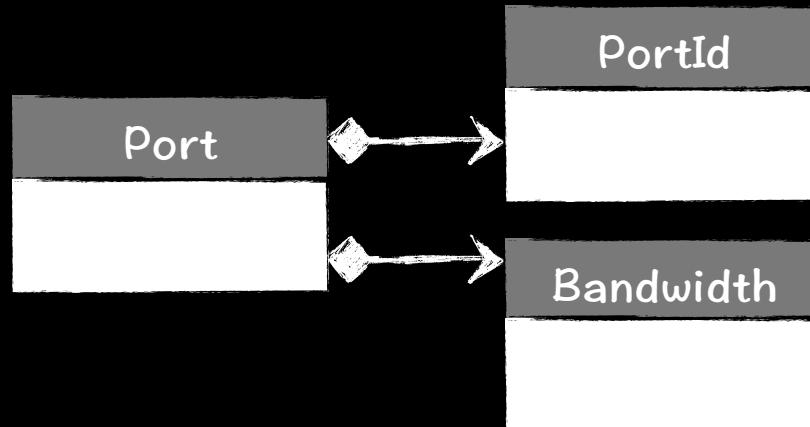
表达概念：命名



命名：清晰简洁，同时表达领域语义与模型角色

- 业务语义：Port, PortId, Bandwidth, Switch…
- 模型角色：xxService, xxEvent, xxFactory, xxRepository, xxContext, xxRole

表达概念：实现选型



- 类 优先于 POD 优先于 基本类型;
- 为基本类型使用领域概念作为别名;
- 为概念赋予行为;
- 单值构造尽量指明“explicit”取消隐式转换;
- 为值对象构造不可变类;

PortId.h

```
using PortId = int;
```

Bandwidth.h

```
#include "cub/base/EqHelper.h"

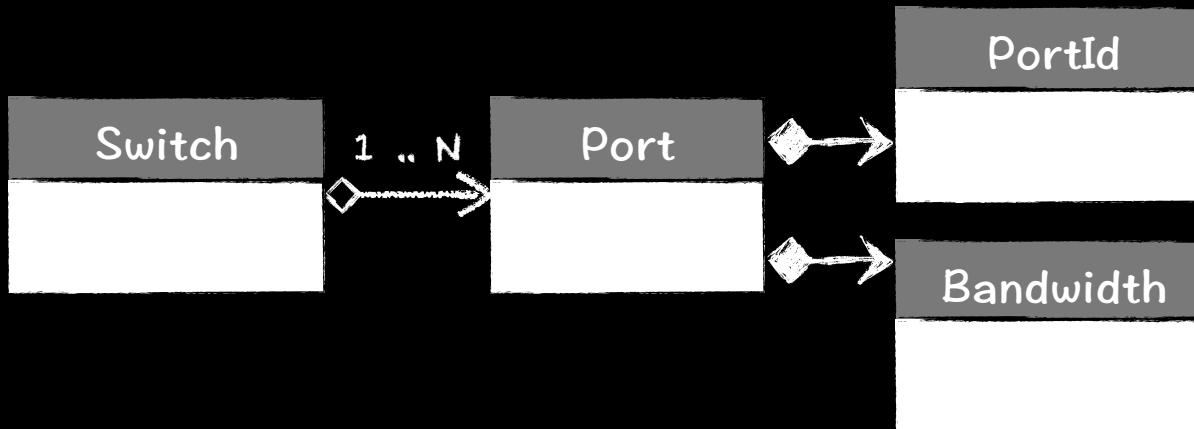
class Bandwidth {
public:
    explicit Bandwidth(unsigned int value)
        : value(value){}

    __DECL_EQUALS(Bandwidth);

    int getExtandFactor() const;

private:
    const unsigned int value;
};
```

表达概念：构造与析构



- 构造与析构服务于对象的生命周期管理；
- 提供完整的初始化语义，不要给状态无意义对象留下空间；
- 显示声明对外部的依存关系和读写约束；
- 表达生命周期早于自己的确定依存时使用引用参数；

Port.h

```

#include "PortId.h"
#include "Bandwidth.h"

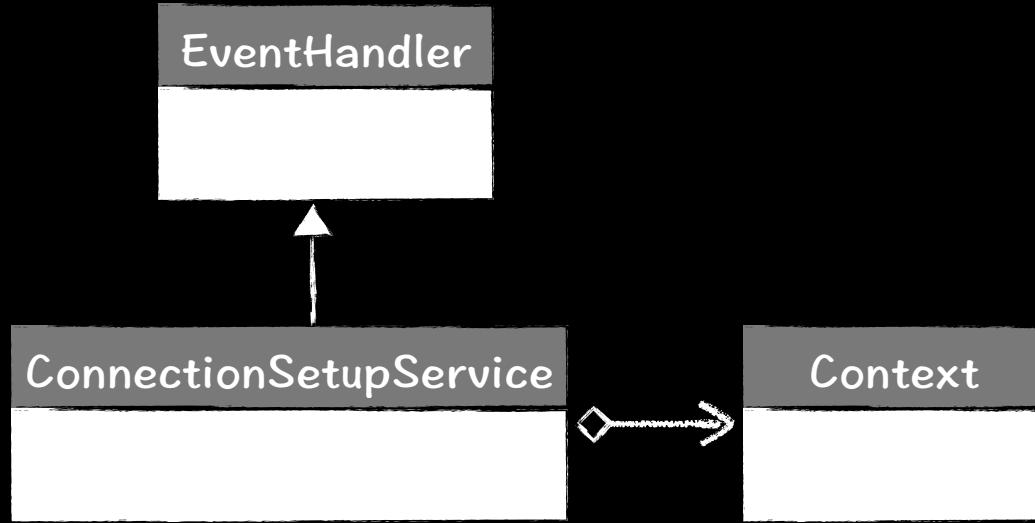
class Switch;
class PeerNode;
class Connection;

class Port {
public:
    Port(PortId id, const Switch& s);
    Connection* connect(const PeerNode& peer);
    void release(const Connection* conn);

private:
    void getBandExpandFactor() const;

private:
    const Switch& switch;
    PortId id;
    Bandwidth bandwidth;
};
  
```

表达概念：行为



- 突出对外API，最小化暴露，尽量少的public；
- 使用const修饰只读方法，遵循CQRS；
- 让接口具有领域含义，尽量不要有get/set（好莱坞原则）；
- 通过公有继承接口，显式化面向客户的声明接口（接口隔离原则）；
- 通过接口访问API，覆写的虚函数private化；

ConnectionSetupEvent.h

```
DEF_INTERFACE(EventHandler) {
    ABSTRACT(Status handle(const ConnectionSetupEvent& event) const);
};
```

ConnectionSetupService.h

```
#include "EventHandler.h"

class Context;

class ConnectionSetupService : public EventHandler
{
public:
    ConnectionSetupService(const Context& ctxt)
        : ctxt(ctxt){}

private:
    OVERRIDE(Status handle(const ConnectionSetupEvent& event) const);

private:
    const Context& ctxt;
};
```

表达概念：属性

Port.h

```
class Port {  
public:  
    Port(PortId id, const Switch& s);  
  
private:  
    PortId id;  
    Bandwidth bandwidth;  
    const Switch& host;  
};
```

Bandwidth.h

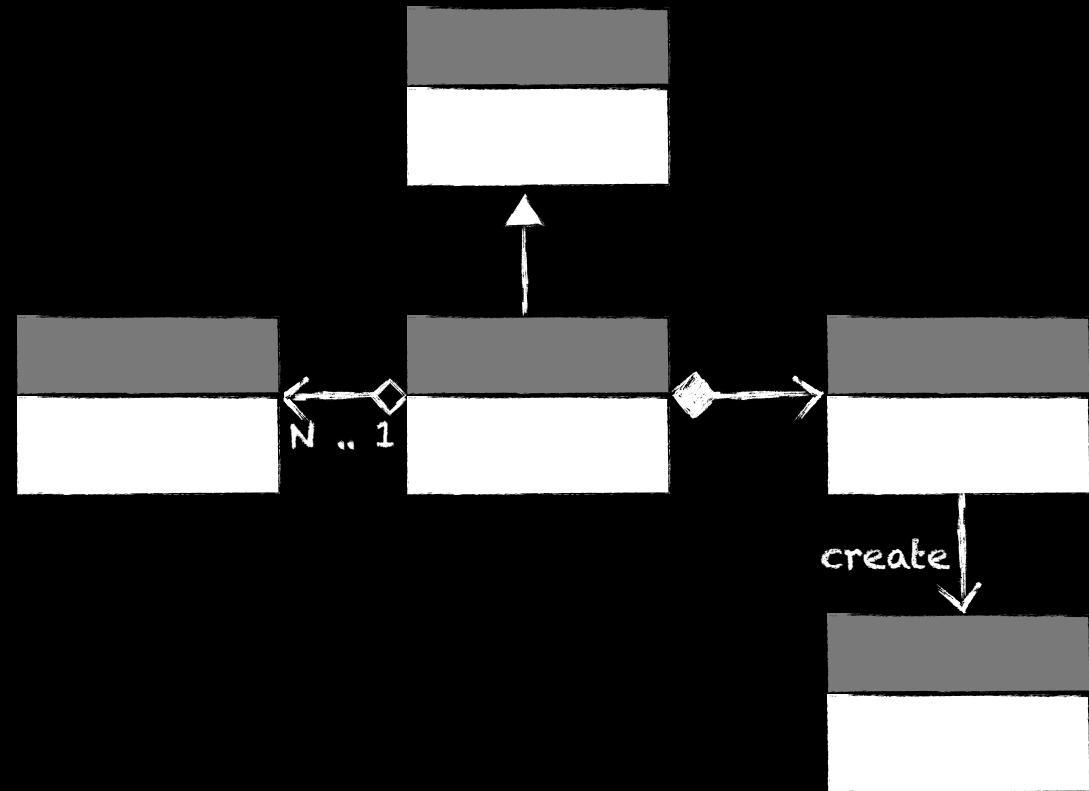
```
class Bandwidth {  
public:  
    explicit Bandwidth(unsigned int value);  
  
    bool operator==(const Bandwidth&) const;  
  
private:  
    const unsigned int value;  
    const unsigned int expandFactor;  
};
```

- 明确的区分属性和依赖；
- 如何表达“我是谁”？为实体的标识和值对象实现比较方法或Has函数；尽量将值对象定义为不可变类；
- 显示化表达依赖关系；（值、指针、引用类型各自表达不同的依赖含义，使用const表达只读）



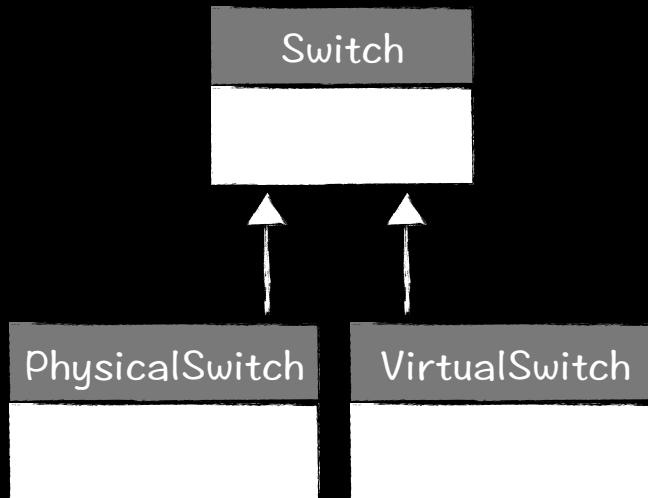
C++实现领域模型的实现模式：表达关系

表达关系



- is-a: 区分实例化与子类化；接口继承与组合；
- has-a: (Composition | Aggregation)，显示化lifecycle关系；
- has-many: 容器选择；实现Repository；依赖关系反转；
- dependency: 不影响存在性；保持物理依赖与逻辑依赖一致；

表达关系：is-a



Switch.h

```

class Switch {
public:
    Switch(unsigned int portNum);
    virtual ~Switch() {};
    Status expandBand();
private:
    ABSTRACT(unsigned int getBandExpandFactor() const);
private:
    unsigned int portNum;
};
  
```

VirtualSwitch.h

```

class virtualSwitch : public Switch {
public:
    virtualSwitch(unsigned int portNum);
private:
    OVERRIDE(unsigned int getBandExpandFactor() const) {
        return 128;
    }
};
  
```

PhysicalSwitch.h

```

class PhysicalSwitch : public Switch {
public:
    PhysicalSwitch(unsigned int portNum);
private:
    OVERRIDE(unsigned int getBandExpandFactor() const) {
        return 64;
    }
};
  
```

Switch.h

```

class Switch {
public:
    Switch(unsigned int portNum,
           unsigned int factor);
    Status expandBand();
private:
    unsigned int portNum;
    unsigned int bandExpandFactor;
};
  
```

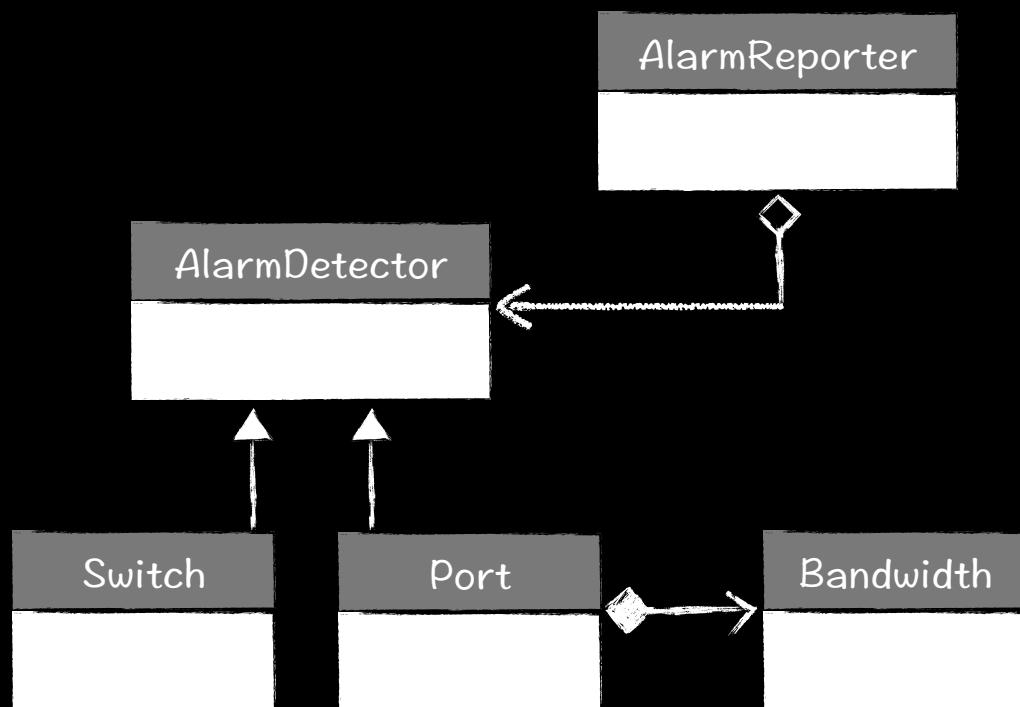
• 数据变化：

数据变化优先使用实例化表达 is-a；

• 行为变化：

行为变化使用多态表达is-a；

表达关系：继承



AlarmDetector.h

```

DEF_INTERFACE(AlarmDetector) {
    ABSTRACT(bool hasAlarm() const);
};

Port.h

class Port : public AlarmDetector
, private Bandwidth {
public:
    Port(PortId id);
    Connection* connect(const PeerNode& peer);

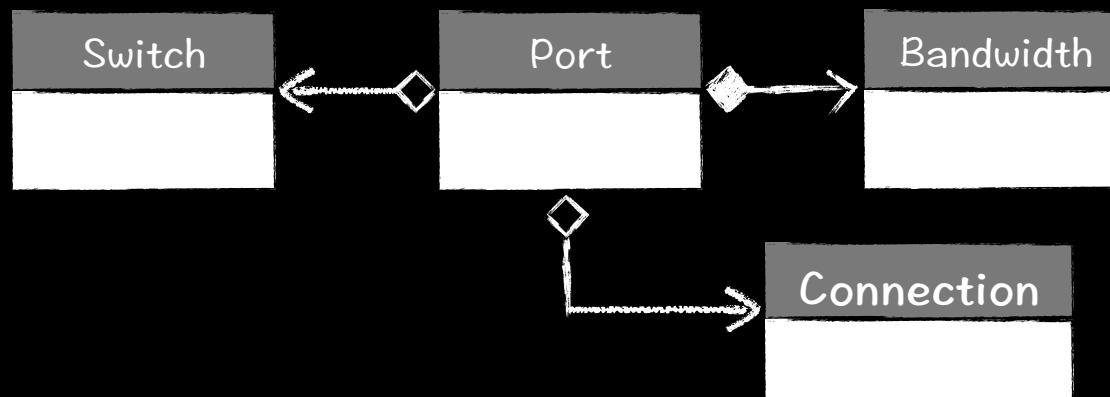
private:
    OVERRIDE(bool hasAlarm() const);

private:
    PortId id;
};
  
```

- 类继承

- 公有继承：继承接口或者角色；客户从上向下使用接口类；覆写方法私有化，禁止从具体类上调用覆写的虚函数；
- 私有继承（以及保护继承）：实际上是一种组合，是生命周期一致的1:1组合关系的一种实现方式；

表达关系：has-a



Port.h

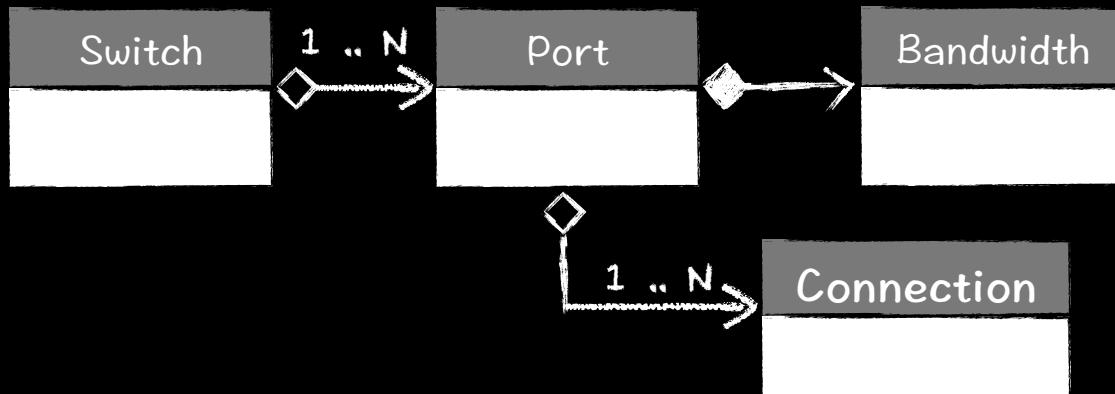
```

class Port : private Bandwidth {
public:
    Port(PortId id, const Switch& s);
    Status connect(const Connection& conn);
    void release(const Connection& conn);

private:
    PortId id;
    Connection* conn;
    const Switch& host;
};
  
```

- 组合（Composition）：优先推荐值包含或者私有继承（默认表达了一致的生命周期）；
- 聚合（Aggregation）：根据生命周期和依存关系，选择使用指针还是引用；用const表示不可更改；
 - 生命周期早于宿主：使用引用，构造函数依赖注入；
 - 生命周期晚于宿主：使用指针；update类接口注入；

表达关系：has-many



Port.h

```

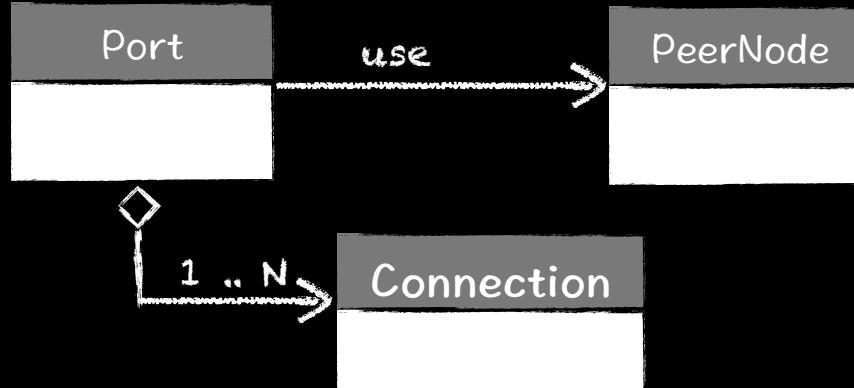
class Port : private Bandwidth {
public:
    Port(PortId id, const Switch& s);
    Status connect(const Connection& conn);
    void release(const Connection& conn);

private:
    PortId id;
    Connection* conn[MAX_CONN_NUM];
    const Switch& host;
};
  
```

- has-many：根据目标需要，实现选择合适的依赖方向

- host -> items: 指针持有，选择合适的容器；
- item -> host : 引用持有，构造注入；
- host <-> items: 双向引用关系，尽量避免；可以让items持有host的一个角色接口；

表达关系：dependency



Port.h

```

class PeerNode;
class Connection;

class Port : private Bandwidth {
public:
    Port(PortId id, const Switch& s);
    Status connect(const PeerNode& peer);
    void release(const PeerNode& peer);

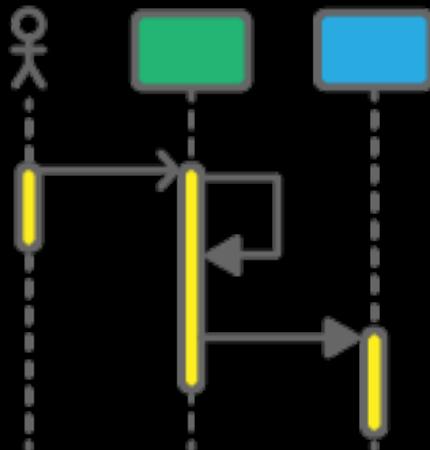
private:
    PortId id;
    Connection* conn[MAX_CONN_NUM];
};
  
```

- dependency：功能使用的需要，非存在性需要；
- 参数依赖，常用于表达use关系，尽可能传递引用类型，使用const表示只读；
- 返回值依赖，常用于表达create关系；一般地，新创建的实体返回指针类型，而新创建的值对象可以返回值类型；
- 减少dependency的物理依赖，尽量前置声明，不要在头文件中包含依赖的头文件；



C++实现领域模型的实现模式：生命周期

生命周期管理



- **生命周期与所有权设计**: 清晰表达领域知识的生命周期和所有权设计;
- **Factory/Repository**: 负责生命周期的管理, 隐藏内存和资源管理细节;
- **嵌入式下特殊的考虑**:
 - 重载**new/delete**: 内存自行管理;
 - **placement** : 内存预占, 延迟初始化; 指定内存构造;
 - **object-pool**: 静态确定对象规格并预占内存;

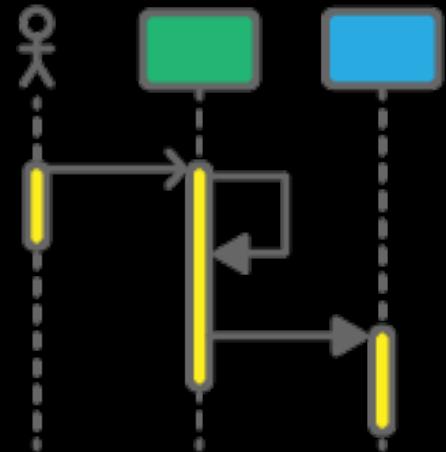
生命周期与所有权设计

• 生命周期

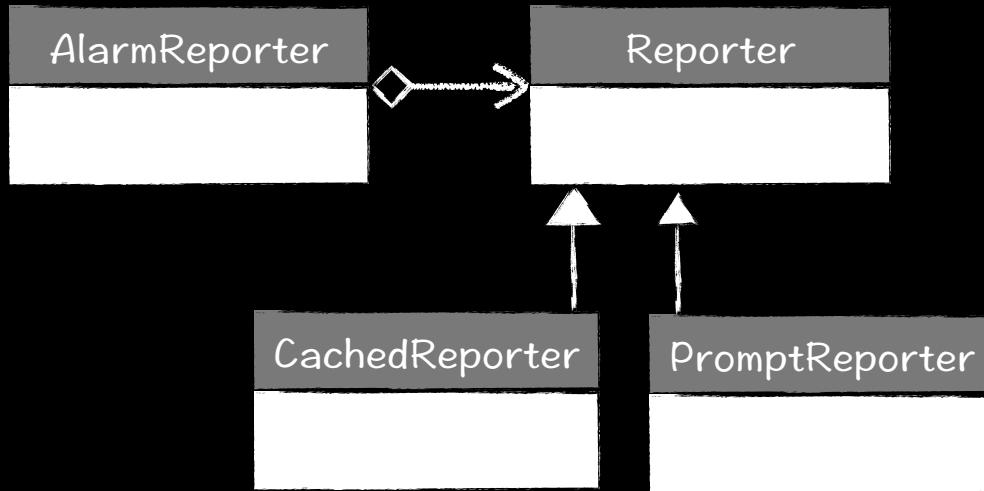
- 拆分不同生命周期的领域对象；
- 清晰的设计领域对象的生命周期：领域对象生命周期开始和结束应该与对应业务时间点保持一致；
- 用factory和repository管理领域对象的生命周期，开始和结束在代码中清晰表达；

• 所有权

- 清晰的领域对象所有权设计，区分所有权与生命周期管理的关系；
- 尽量不共享所有权；明确领域对象所有权的权威持有者，尽量只共享只读所有权；
- 用unique_ptr表达同时持有对象的所有权和生命周期管理，只能转移，不能分享；
- shared_ptr共享了所有权，让生命周期管理隐式化。可以用来管理生命周期动态化对象，谨慎使用；
- weak_ptr表示一种弱持有，只存储一个对象的可访问句柄，不负责生命周期管理；每次使用前需要判断有效性；
- 靠设计简化所有权和生命周期管理，例如如果树生成时先有根后有叶子（释放时相反），那么根持有叶子的unique_ptr，叶子持有根的裸引用即可；



placement



• placement

- 在指定的内存上构造对象；
- 按照系统字节对齐要求预占内存；
- 为必须有参构造的对象及数组预占内存；
- 和union结合，在指定内存上实现静态多态；

Reporter.h

```

DEF_INTERFACE(Reporter) {
    ABSTRACT(Status report(const AlarmInfo&));
}
  
```

CachedReporter.h

```

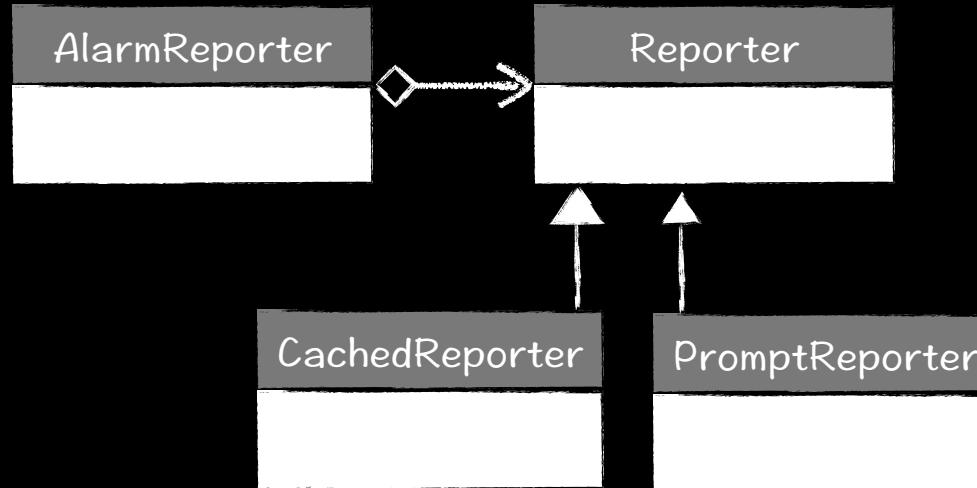
class CachedReporter : public Reporter {
public:
    CachedReporter(const Address& addr);
private:
    OVERRIDE(Status report(const AlarmInfo&));
}
  
```

PromptReporter.h

```

class PromptReporter : public Reporter {
public:
    PromptReporter(const Address& addr);
private:
    OVERRIDE(Status report(const AlarmInfo&));
}
  
```

placement



- placement

- 在指定的内存上构造对象；
- 按照系统字节对齐要求预占内存；
- 为必须有参构造的对象数组预占内存；
- 和union结合，在指定内存上实现静态多态；

AlarmReporter.h

```

class AlarmReporter {
public:
    enum ReporterType {
        CACHED,
        PROMPT,
    };

    AlarmReporter(ReporterType type, const Address& addr)
        : reporter(nullptr) {
        if(type == CACHED) {
            reporter = new (u.cached.alloc()) CachedReporter(addr);
        } else {
            reporter = new (u.prompt.alloc()) PromptReporter(addr);
        }
    }

private:
    union {
        Placement<CachedReporter> cached;
        Placement<PromptReporter> prompt;
    } u;

    Reporter *reporter;
};

```

object-pool

- 嵌入式下factory/repository的实现模式

- 对象重载new/delete;
- factory单例，在实现文件内隐藏内存池；
- 预先规划对象规格，静态内存分配；
- repository用容器管理factory产生的对象指针；



PortFactory.h

```

DEF_SINGLETON(PortFactory) {
    Port* createPort(PortId id, const Switch&) const;
    void releasePort(Port*) const;
};
  
```

Port.h

```

class Port {
public:
    Port(PortId id, const Switch& s);
    // ...
    DECL_OPERATOR_NEW();
private:
    // ...
};
  
```

PortFactory.cpp

```

namespace{
    DEF_OBJ_ALLOCATOR(Port, 16);
}

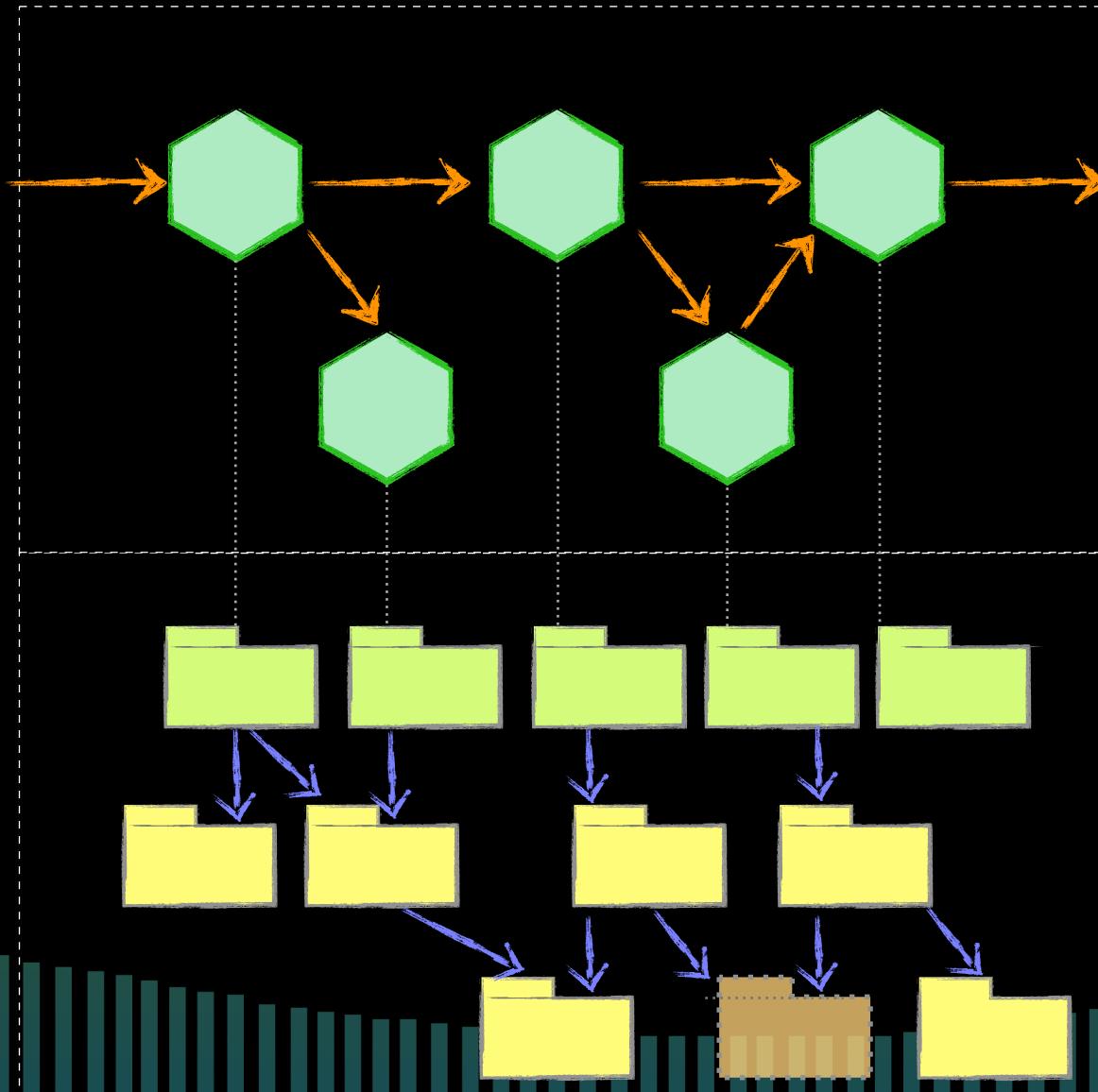
Port* PortFactory::createPort(PortId id, const
Switch& switch) const {
    return new Port(id, switch);
}

void PortFactory::releasePort(Port* port) const {
    delete port;
}
  
```



C++实现领域模型的实现模式：物理设计

逻辑依赖和物理依赖



- **逻辑依赖:** 表示Is-a, Has-a, Depends等逻辑关系，强调系统内逻辑概念之间的依存和交互关系；
 - 逻辑依赖先于物理依赖；
 - 勿从物理视图和实现推证逻辑依赖合理性，应正向领域建模；
 - 用现代IDE辅助展现逻辑结构和依赖；
-
- **物理依赖:** 表示头文件/实现文件、目录和包的物理组织关系，包括编译时依赖和链接时符号依赖；糟糕的物理设计导致构建低效、复用困难，架构隐晦。
 - 物理设计需要考虑构建系统和发布系统的约束，同时尽量保持与逻辑设计的一致性；
 - 借助包管理等工具自动维护和展现物理依赖关系视图；

物理设计：符号依赖

减少符号依赖：

- 尽量避免使用“**extern**”关键字，除非引用某些汇编或者编译器生成符号；
- 尽量使用“**static**”关键字隐藏符号；
- C++尽可能多使用**private**和匿名命名空间隐藏符号；
- 对于**弱符号**（指针/引用，函数参数类型，返回值类型）使用前置声明而非包含头文件；
- 使用**PIMPL手法**，将头文件中无需暴露的细节转移到实现文件中；

物理设计：头文件设计

头文件设计原则：

- **自满足原则：**头文件要独立可编译通过（头文件通过放到自身实现文件第一行可自行验证自满足性）；
- **最小依赖原则：**头文件在可独立编译的前提下基础上尽可能少依赖其它头文件和符号（尽量弱符号依赖）；
- **最小可见原则：**头文件暴露最小信息，不让客户看到不需要的信息；
 - 将客户无需看到的宏、类型、静态私有成员和函数搬迁到实现文件；
 - 对头文件中的接口进行可见性管理（包外、包内）
 - 减少头文件中不必要的inline函数；
 - 适可使用PIMPL技巧隐藏更多的头文件中信息；
- **接口隔离原则：**
 - 把明确的面向不同客户的接口，拆分到不同头文件中，避免不同客户因为共享头文件产生耦合；避免上帝头文件；

物理结构设计

Port

- id : PortId
- bandwidth : BandWidth

- + Port(id : PortId, s : Switch)
- + connect(peer : PeerNode) : Connection
- + release(conn : Connection)

Port.h

```
#include "PortId.h"
#include "Bandwidth.h"

class Switch;
class PeerNode;
class Connection;

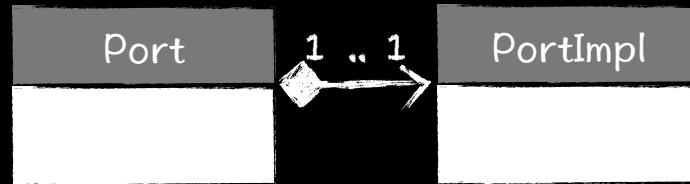
class Port {
public:
    Port(PortId id, const Switch& s);
    Connection* connect(const PeerNode& peer);
    void release(const Connection* conn);

private:
    void getBandExpandFactor() const;

private:
    PortId id;
    Bandwidth bandwidth;
};
```

- 一个.h和.cc只表达唯一一个概念；
- 保持文件和模型元素的命名一致性；
- 头文件接口： 依赖 -> 接口 -> 属性的顺序；
- 使用前置声明减少依赖；
- 遵循自满足，最小依赖原则，最小可见原则；

C++ PIMPL模式



Port.h

```
class PeerNode;
class Connection;
class PortImpl;

class Port {
public:
    Port(PortId id, const Switch& s);
    Connection* connect(const PeerNode& peer);
    void release(const Connection* conn);
    ~Port();
private:
    PortImpl* pimpl{nullptr};
};
```

Port.c

```
#include "Port.h"
#include "PortId.h"
#include "Bandwidth.h"
#include "Switch.c"

namespace {
    class PortImpl {
public:
    PortImpl(PortId id, const Switch& s);
    Connection* connect(const PeerNode& peer);
    void release(const Connection* conn);

private:
    void getBandExpandFactor() const;

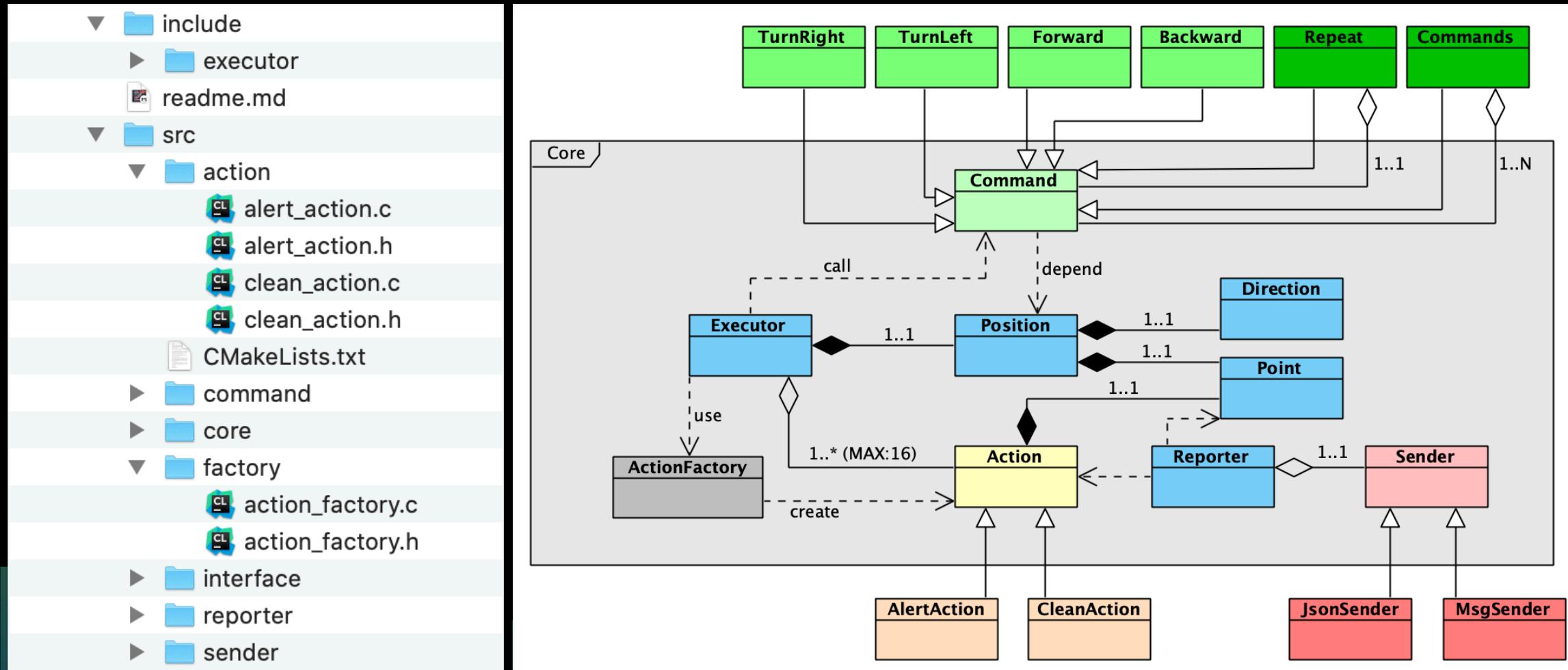
private:
    PortId id;
    Bandwidth bandwidth;
    const Switch& switch;
};

Port::Port(PortId id, const Switch& s) {
    this->pimpl = new PortImpl(id, s);
}
Connection* Port::connect(const PeerNode& peer) {
    return this->pimpl->connect(peer);
}
```

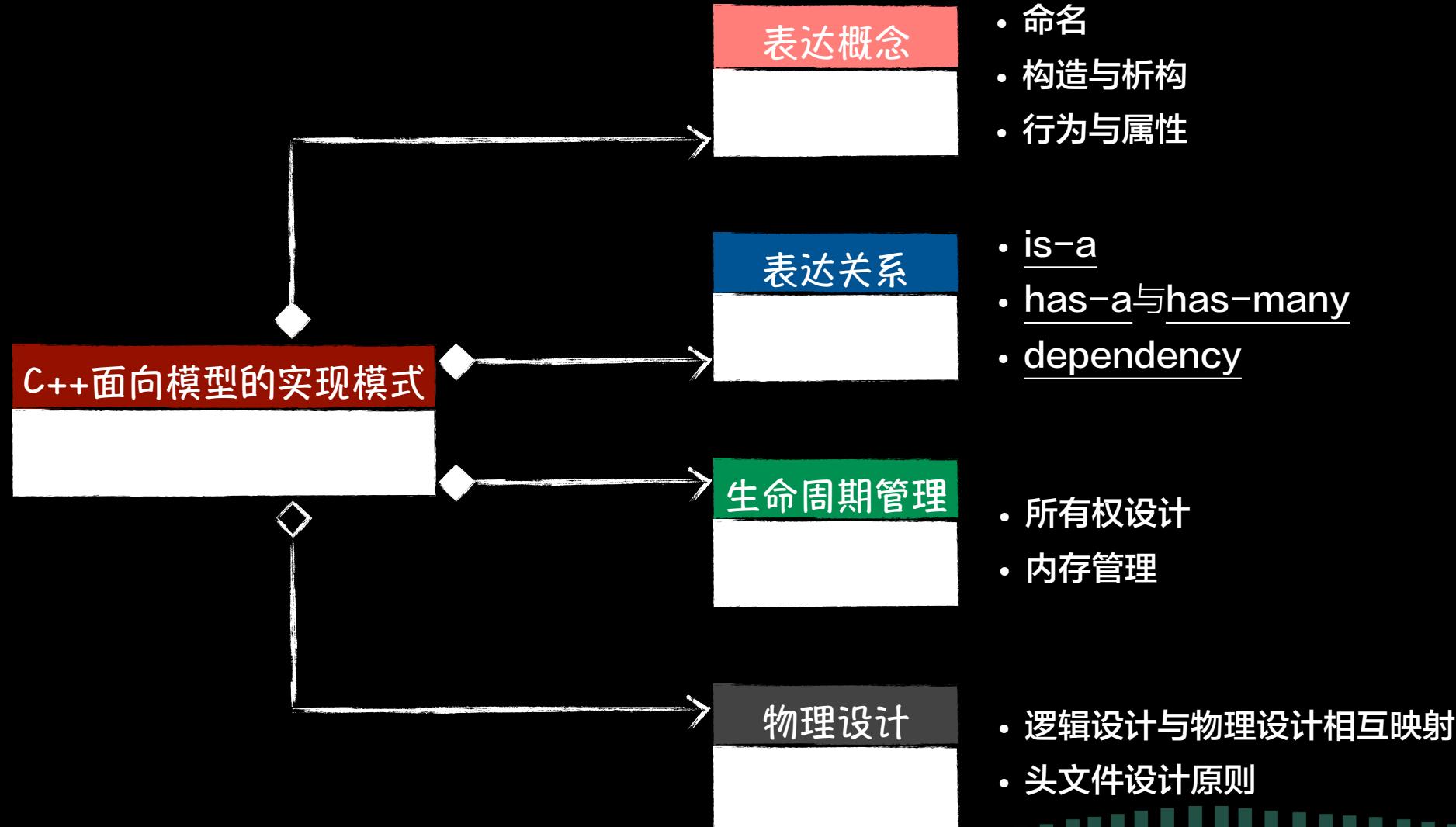
- 可以使用PIMPL模式，消除头文件耦合和物理化问题

物理设计：目录设计

- **按照不同的稳定度和复用度划分**: 例如按照核心部分、接口部分以及每个独立扩展方向进行目录划分;
 - **目录结构优先关注构建和发布约束, 兼顾逻辑结构一致性**;



总结



谢谢大家！

刘新铭

鉴释科技首席架构师，编译器专家



刘新铭是国际上少数精通编译器技术的计算机科学家之一，曾担任惠普Java编译器技术实验室主任，领导基于惠普安腾处理器的编译器开发工作。他在计算机语言设计和高级编译器优化技术方面具有深入的实践经验。迄今为止，在程序分析和优化领域获得了十余项技术专利，而且在多个核心技术期刊上发表了多篇重量级论文。2018年联合创立了鉴释公司。

主办方：

Boolan
高端IT咨询与教育平台

李隆

鉴释科技首席科学家



目前担任鉴释科技核心技术的首席科学家，专注于代码验证基础架构。于2008年在中科大获得计算机软件和理论博士学位，学术研究包括证明基于语言的分析技术在构建高效且可靠的软件方面的应用，并发表了数篇期刊和会议论文。毕业后，李隆博士加入了三星电子，从事高级技术小组的统计机器翻译工作。并于2010年加入HP编译器团队，从事HP Non-Stop编译器后端和SDK。

主办方：

Boolan
高端IT咨询与教育平台



鉴 释

xcalibyte

A TOOL FOR VERIFYING BUSINESS LOGIC
BEYOND COMMON VULNERABILITIES

WHO ARE WE?

SHIN-MING LIU 刘新铭
鉴释首席架构师

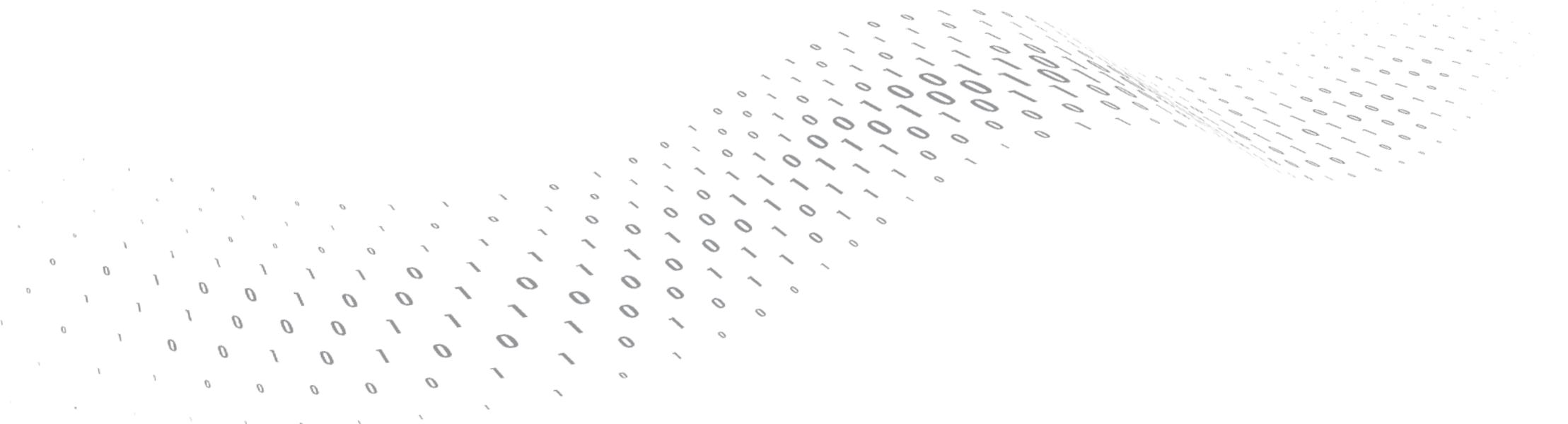


- Compiler Scientist
- Director China Intel IOT Research Lab
- Director HP Compiler Technology Lab
- 10+ Patents granted in program analysis and compiler optimization

LI LONG 李隆
鉴释首席科学家



- PhD in CS, USTC, Software Security Laboratory
- 8+ Years HP NonStop compiler backend & SDK engineer



*Xcalibyte's mission is to improve
the quality of software by creating
easy-to-use tools that help
developers build & deploy reliable
& secure code*

WHY ARE WE HERE?

- ✓ IT Technology is at a turning point with:
 - **Domain Specific Hardware** e.g. AI which will be pervasive
 - Two decades of a **software boom** which needs to be sustained
 - Ubiquitous **distributed computing** in a connected world
- ✓ Software Challenges include:
 - Bugs occurring at greater frequency
 - 82% of security issues are from applications
 - 1 in 1000 lines of code having security issues
 - 1 in 1400 lines of code having high severity security issues

Xcalibyte, is here to help!



WHERE DO SECURITY ISSUES COME FROM?

- ✓ Vulnerabilities incubated in applications that are exposed
- ✓ Violations of underlying business logic