Robust Distributed System Nucleus (rDSN)

https://github.com/Microsoft/rDSN

Zhenyu Guo (@imzhenyu)

Building robust distributed systems is important

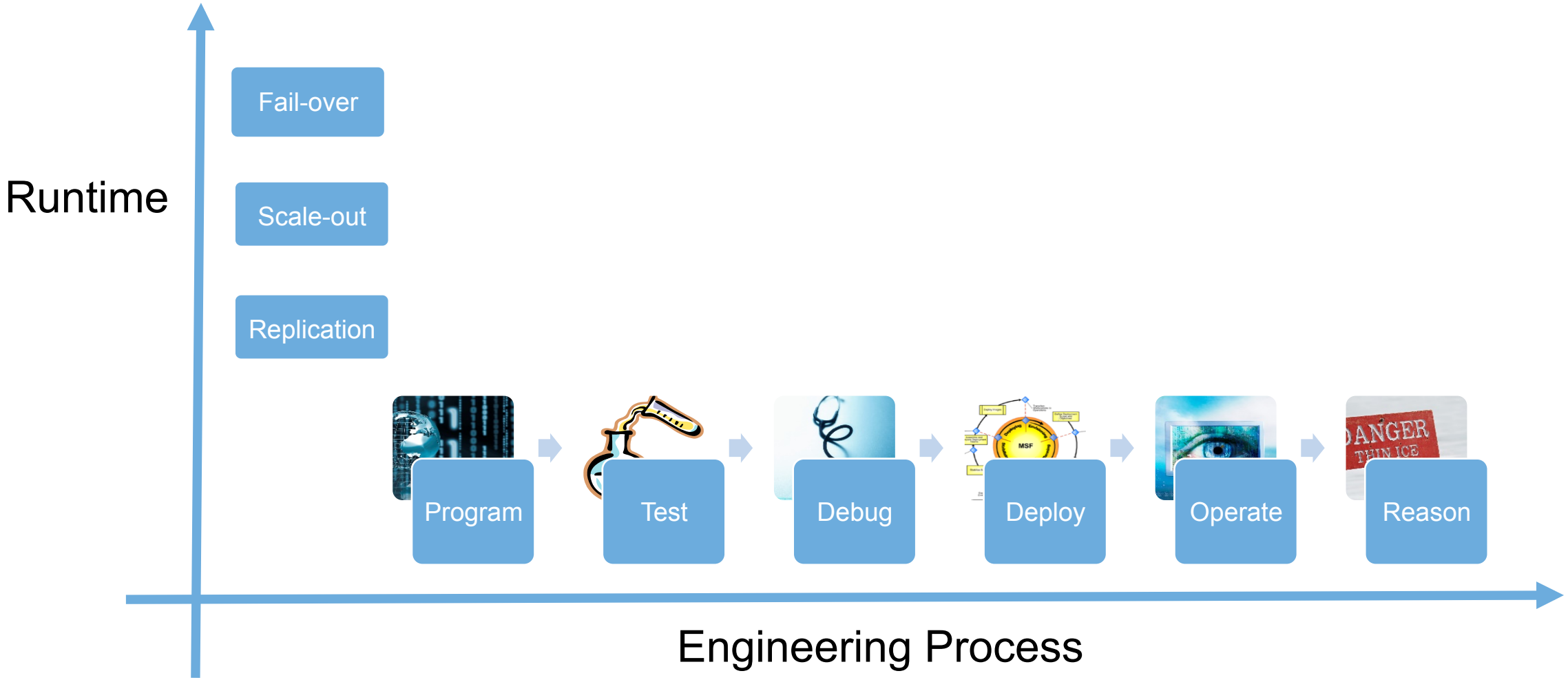
- Our lives are relying on distributed (computer) systems more and more
  - (Traditional) Reading, Entertainment, Communication, ...
  - (O2O) Taxi, Dating, Tickets, Doctor, ...
  - (IoT) Health/Security Monitoring, Smart Home/Car, IFTTT, …
  - …

- Alipay was offline for ~2 hrs starting 2015.5.27 5PM[1]
  - @客厅之家：还要不要人愉快的卖家具了？刚卖出就无法付款，万一不买了，谁来承担啊?
  - @华南女院：我吃饭没带钱包和银行卡，想说用支付宝付款的。这顿饭已经假装吃了一个多小时了！尼玛！！
  - …

1.http://money.hexun.com/2015-05-29/176281908.html

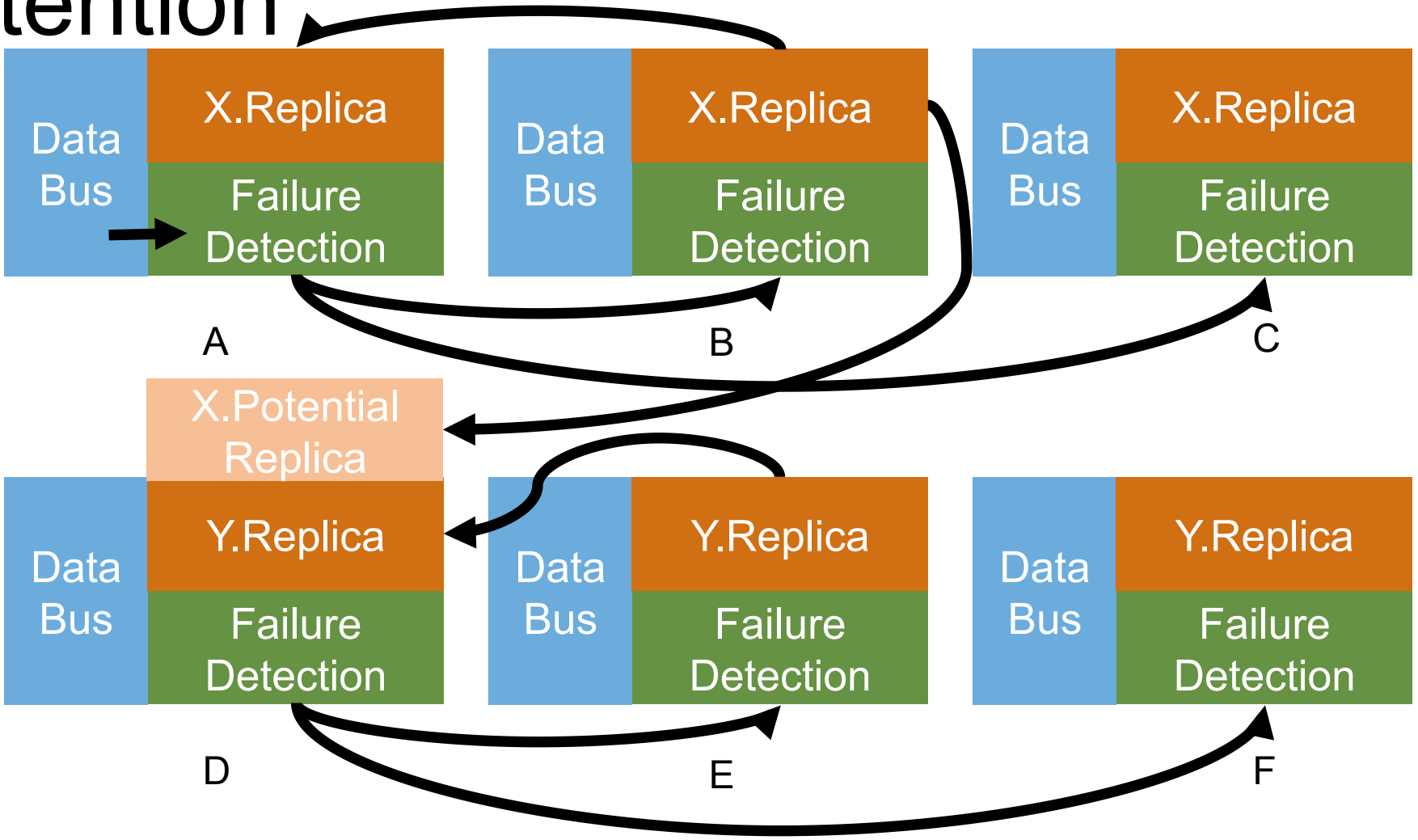
Service robustness is far from being perfect

Many works done to enhance robustness



However, many are ineffective, not adopted, or even doing the opposite!

Example: replication w/ resource contention



Try to enhance robustness with automated failure recovery, but amplify a small cold into a contagious deadly plague!

Ex

Problem:
that ma

Case study: Slice-based error log analysis using a simple query

- Automatic focus on relevant log entries
- Declarative analysis without headaches of distributed systems

Related Logs

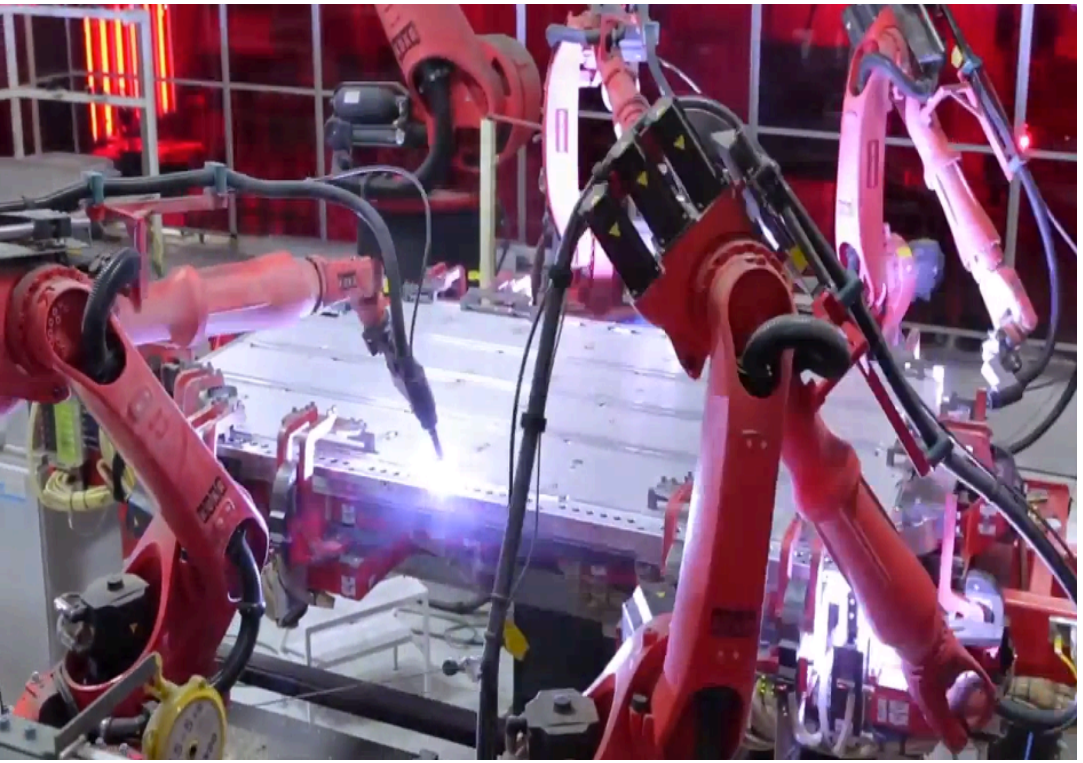
```
Events.Where(e => e.Type == Event.ErrorLog
                  && e.Payload.Contains("02/24/2009 07:20:27"))
      .Slicing(Slice.Backward)
      .Select (s => s.DumpHtml("AssertionLog"))
      .Count();
```

Backward Slicing

What we do to enable a given system for this tool?

- Redirect all the logs
- Instrument all the asynchronous execution and messages flows
  - (**Incomplete**) Difficult to be complete (usually 10+ even 20+)
  - Sometimes **unsafe** (hidden assumption)
  - **Fragile** due to later system development
- Also required by other tools/frameworks, e.g., automated test, replay, replication, illustrated later

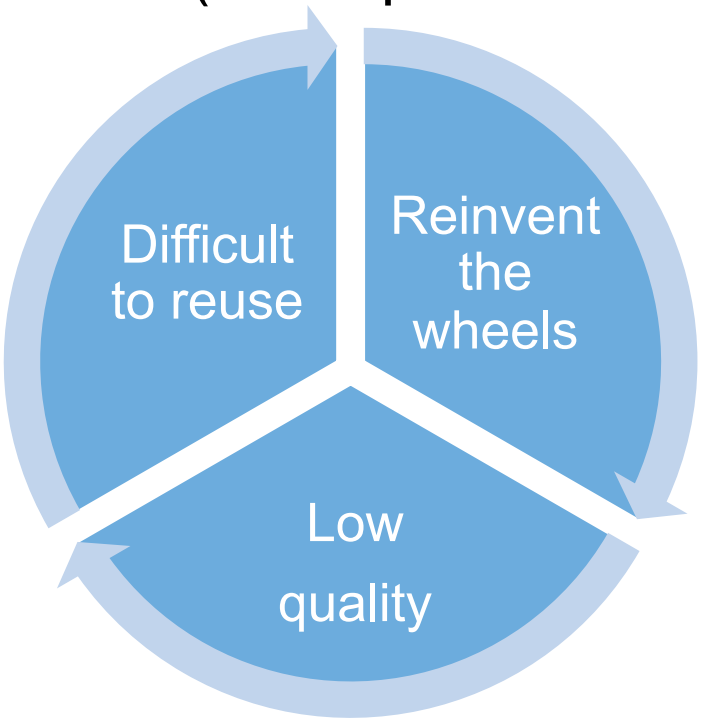
So what we see today



VS

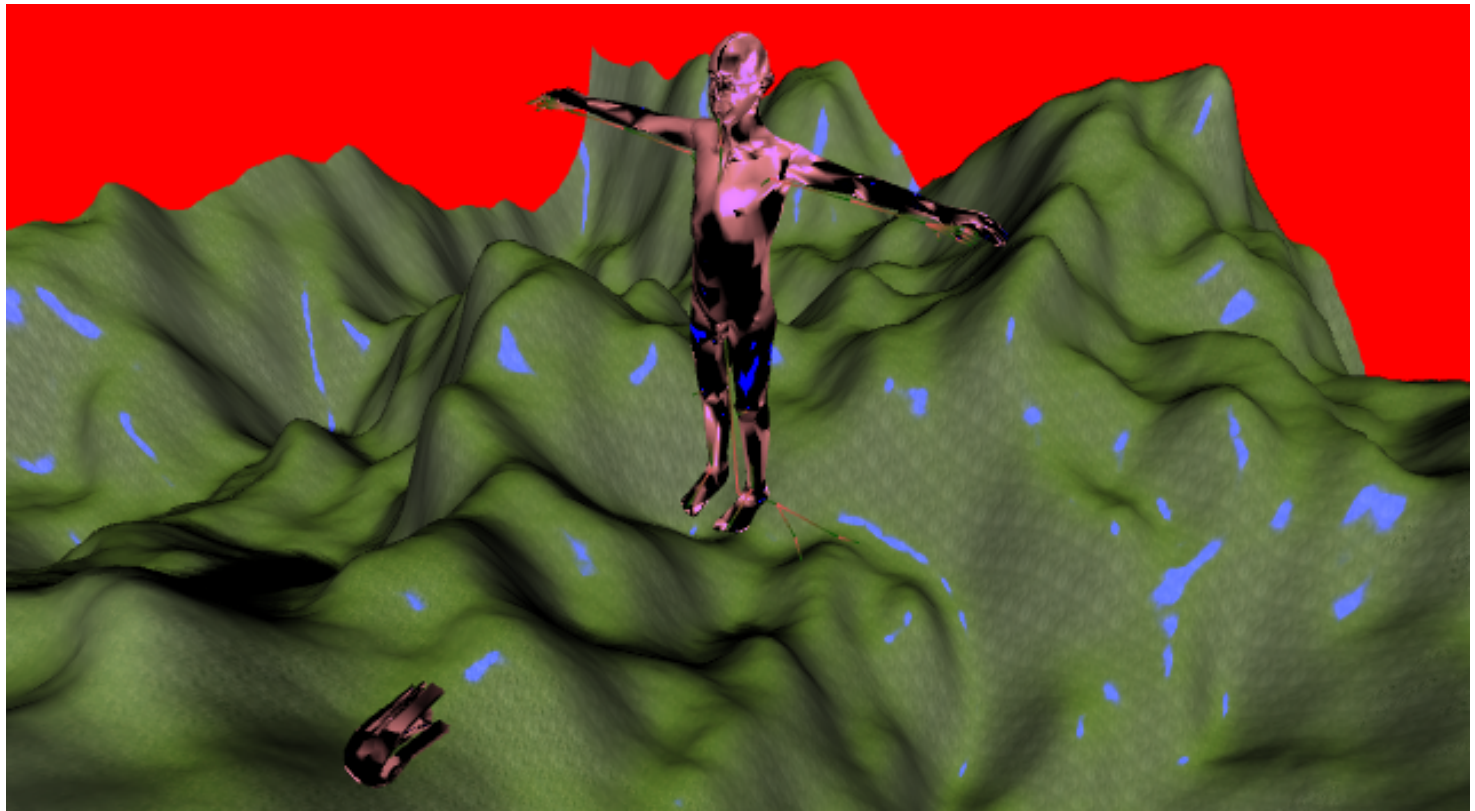
What are the problems?

- Implicit contract
  - Hidden/volatile dependencies/assumptions
  - Unknown interference
  - Systems remain black boxes therefore difficult (incomplete/unsafe/fragile) to build automation and robustness



Problems (2)

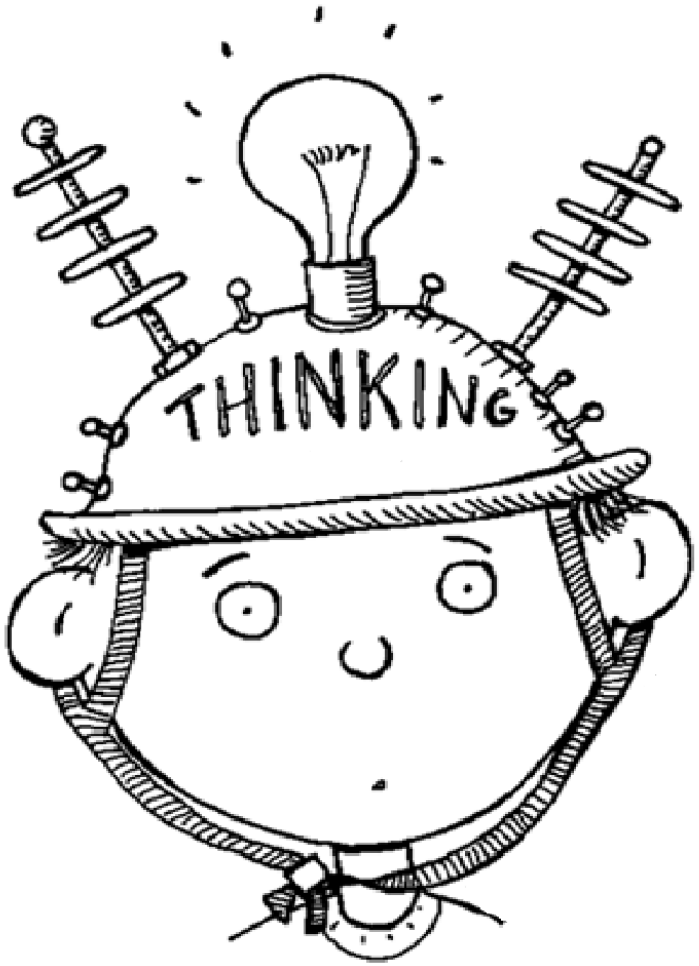
- Lack of `Systems Thinking'
  - Local and global decisions do not align which leads to robustness issues
  - e.g., resource contention, local failure recovery



Problems (3)

- Intrusive (After-thought) tooling support
  - Testing, debugging, and post-mortem analysis is difficult

Seems difficult to fix for the legacy code - what if we come up a new development framework?



Design goal

- Turn distributed systems from black boxes into white ones
  - Explicit dependency, non-determinism, and interference
  - At various granularities (e.g., task, module, service, …)
- Embrace Systems Thinking
  - Break the dilemma between modularity and global coordination
- Native tooling support
  - Build first-class tooling support into both system runtime and engineering process in a coherent way
- Inclusive
  - Support legacy components/languages/protocols/environments as much as we can

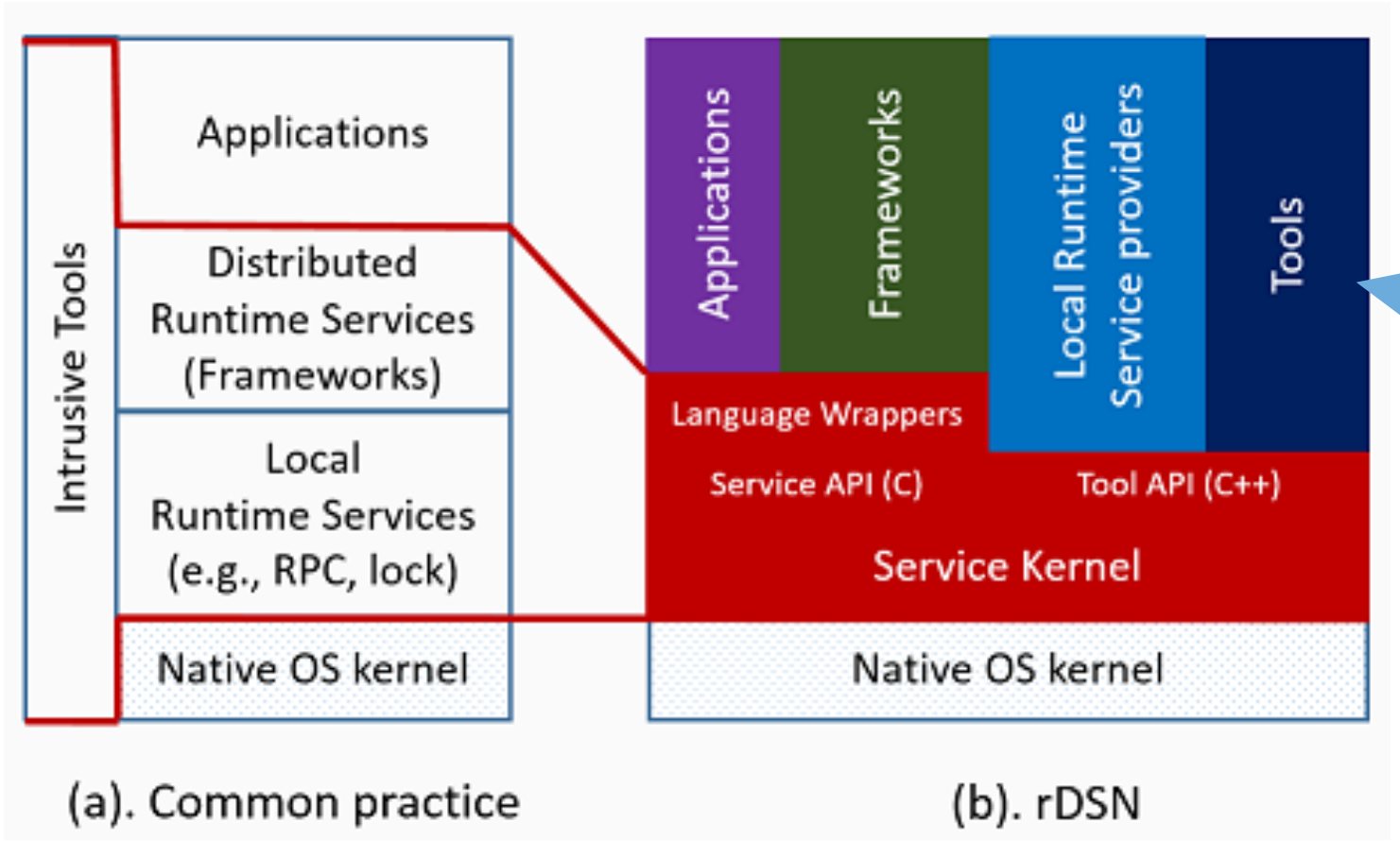
Robust Distributed System Nucleus (rDSN)

- Robustness is at the core of its design
- A cloud service development framework with pluggable modules
  - Applications => business logic
  - Frameworks => distributed system challenges
  - Devops tools => engineering and operation challenges
  - Local runtime/resource libraries

- Existing and growing modules benefiting each other
- An ecosystem for quickly building robust distributed systems

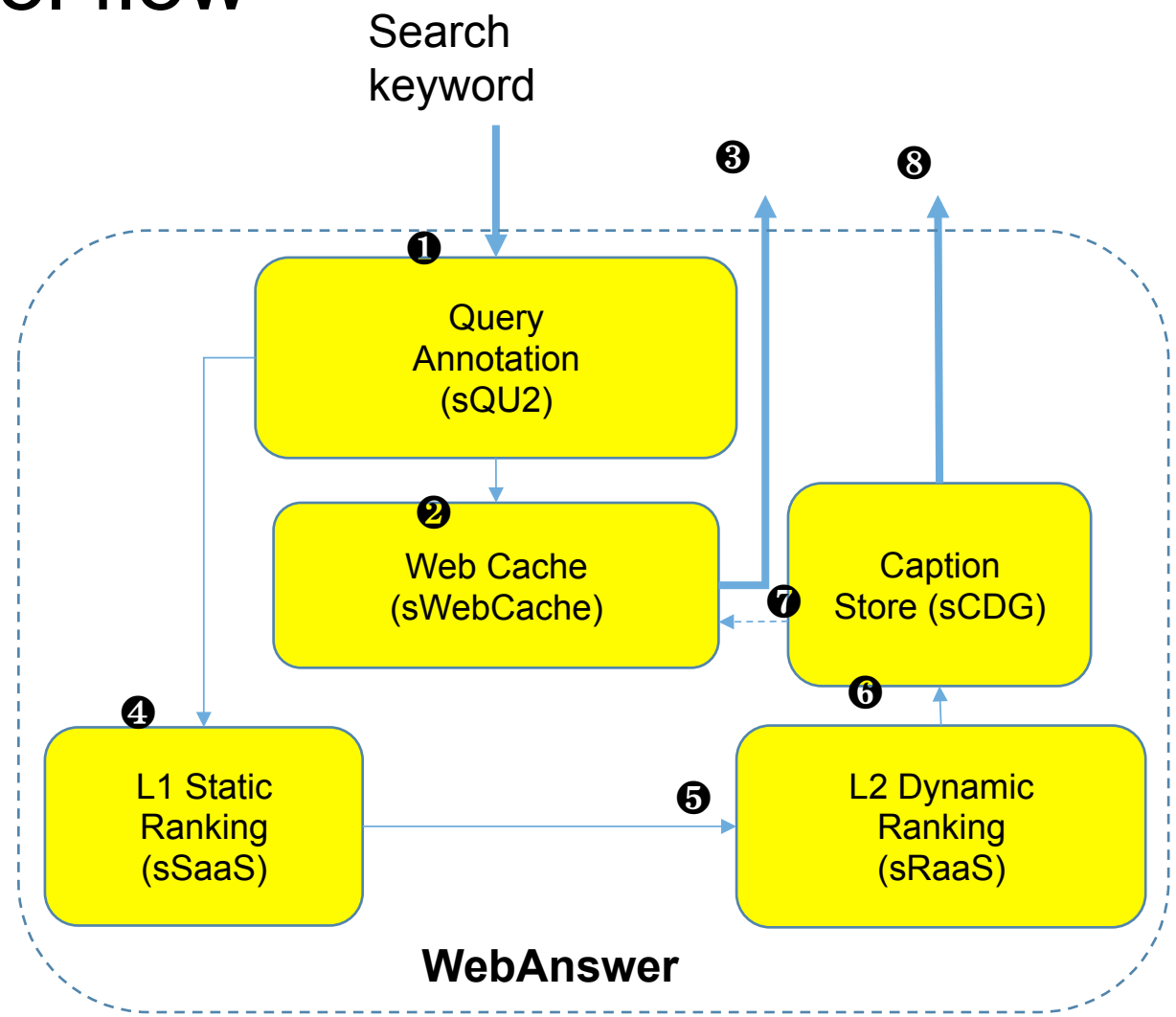
Key Design Decisions

Explicit contract via microkernel architecture

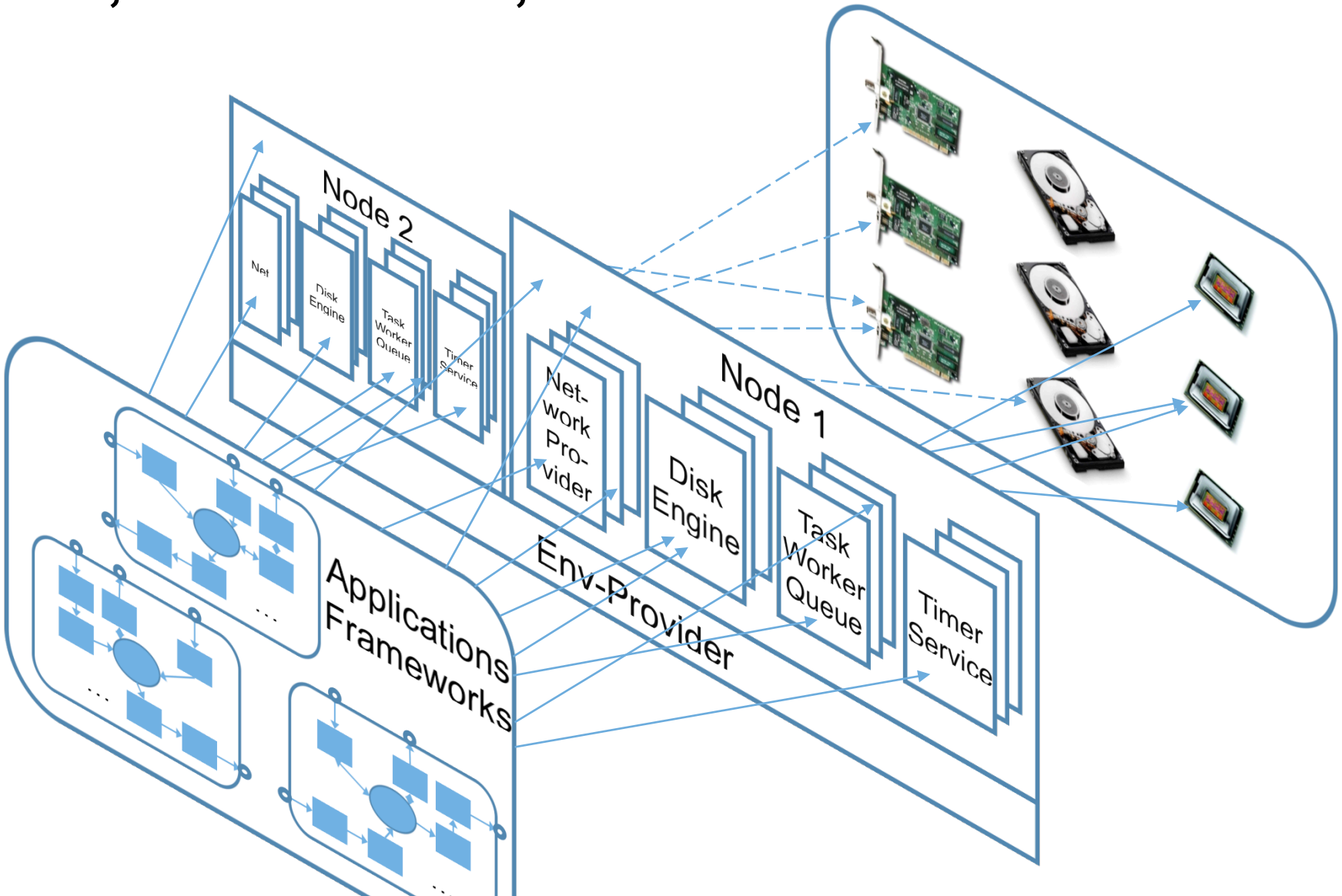


NO direct interaction among the frameworks, applications, tools, and local libraries, to ensure dependencies and non-determinisms reliably captured and manipulated.

Explicit cross-service contract managed though declarative data and control flow



Late binding with deploy-time global view for modules, resources, and their connections



First class tooling support

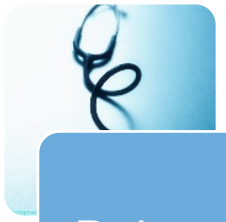
- Dedicated tool API for fast tool development
- Transparent and reliable integration with ALL upper apps and frameworks

Program

Code generator
…

Test

Fault injection
Model checking
Global assertion
Declarative
Scenario test
…



Debug

Replay
Simulator

Perf Tuning

Bottom-up analyzer
Profiler
…

Deploy

Declarative deployment model,
App store,
Auto-deployment
…

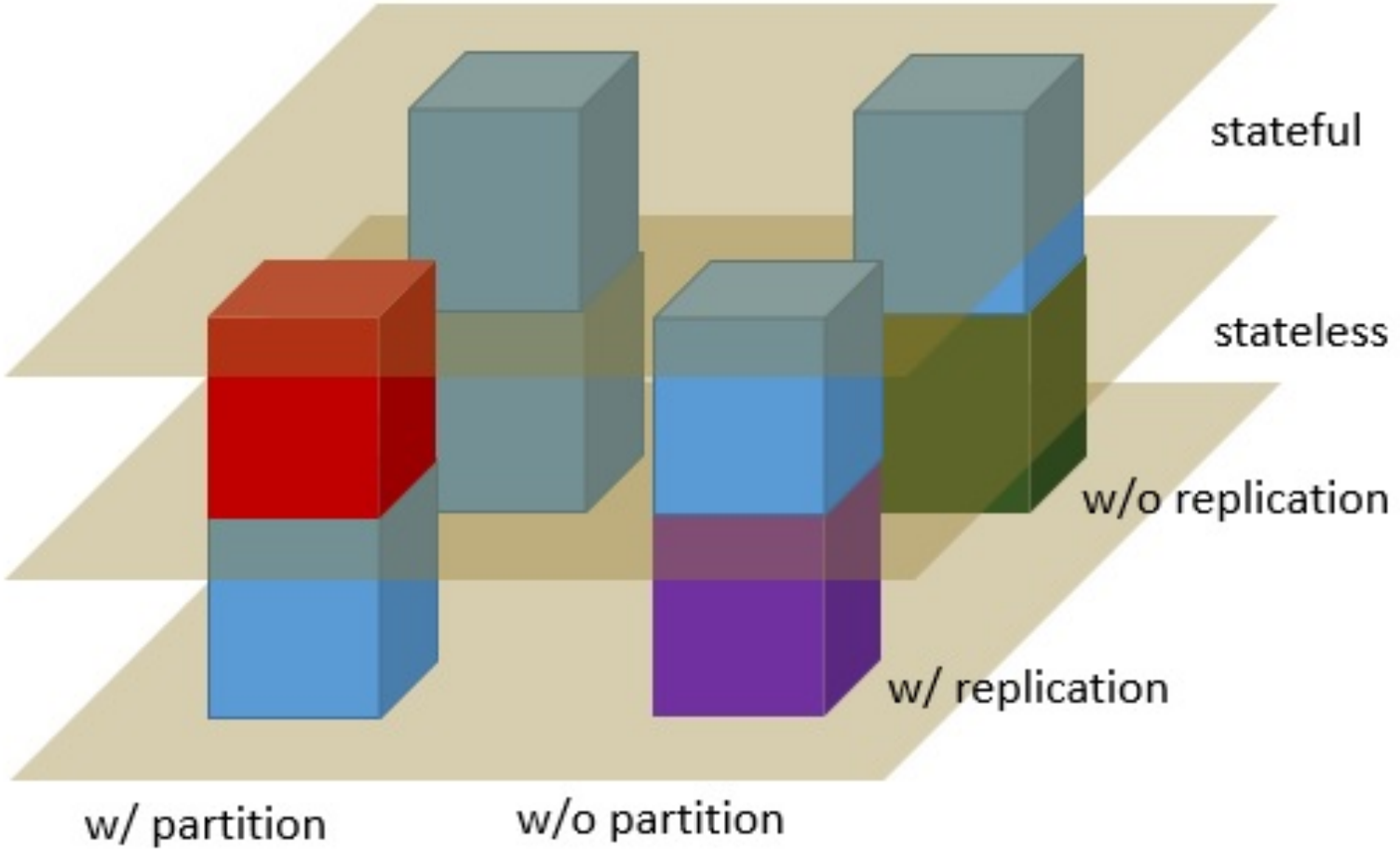
Operate

Dashboard,
Tracer,
Profiler
…

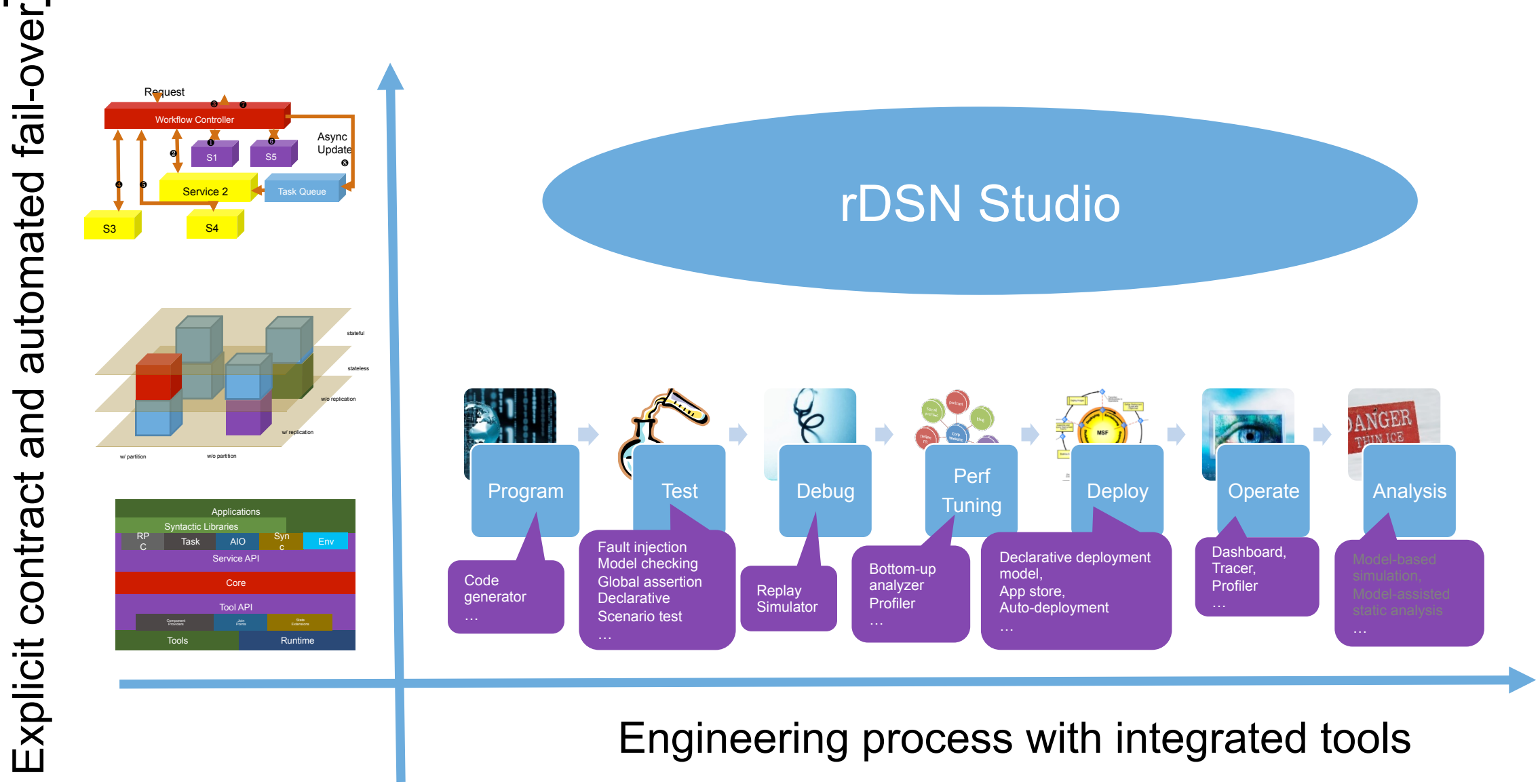
Analysis

Model-based simulation,
Model-assisted static analysis
…

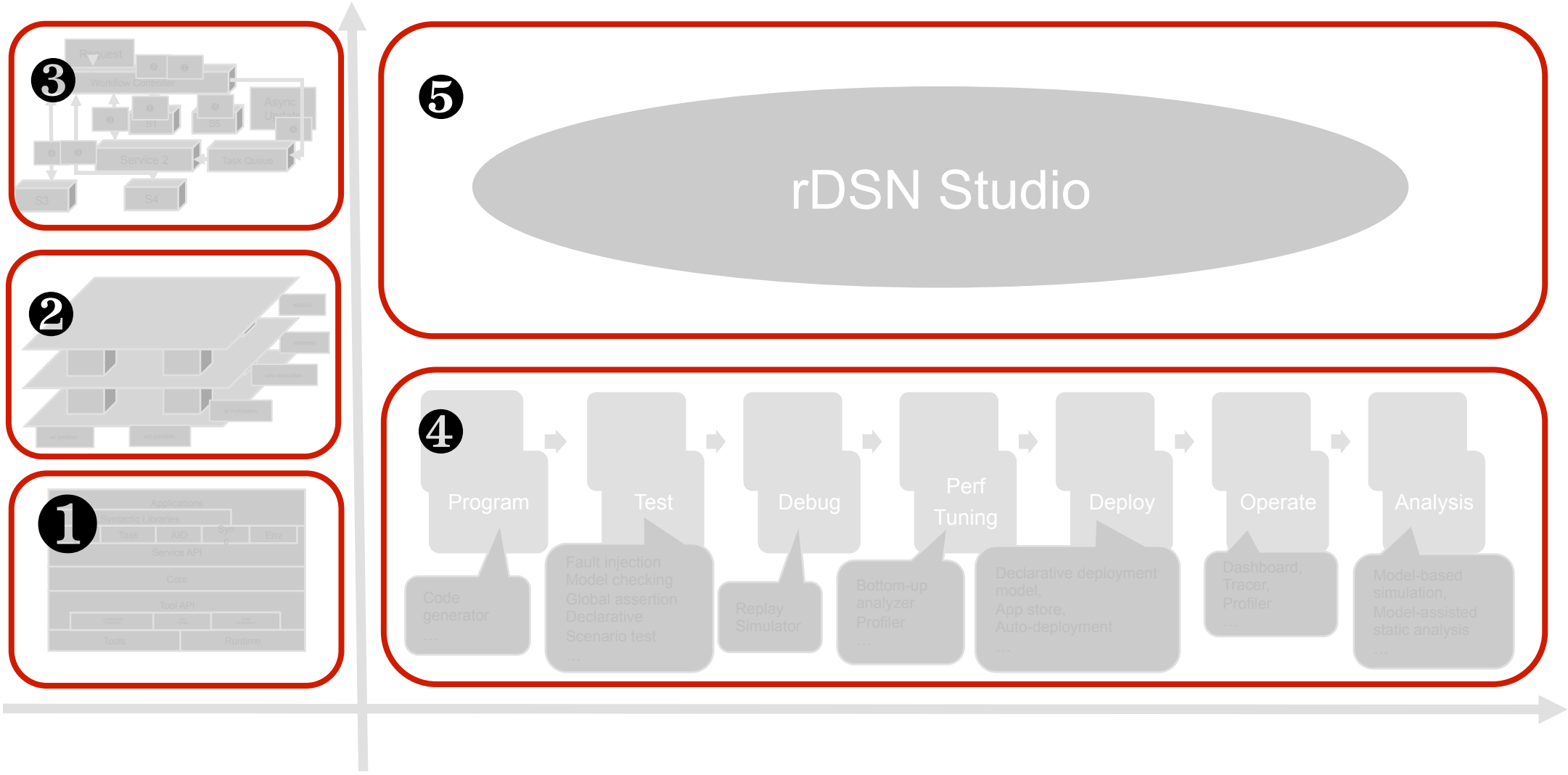
Common service model for auto scale-out and fail-over for ALL upper applications (micro service and storage)

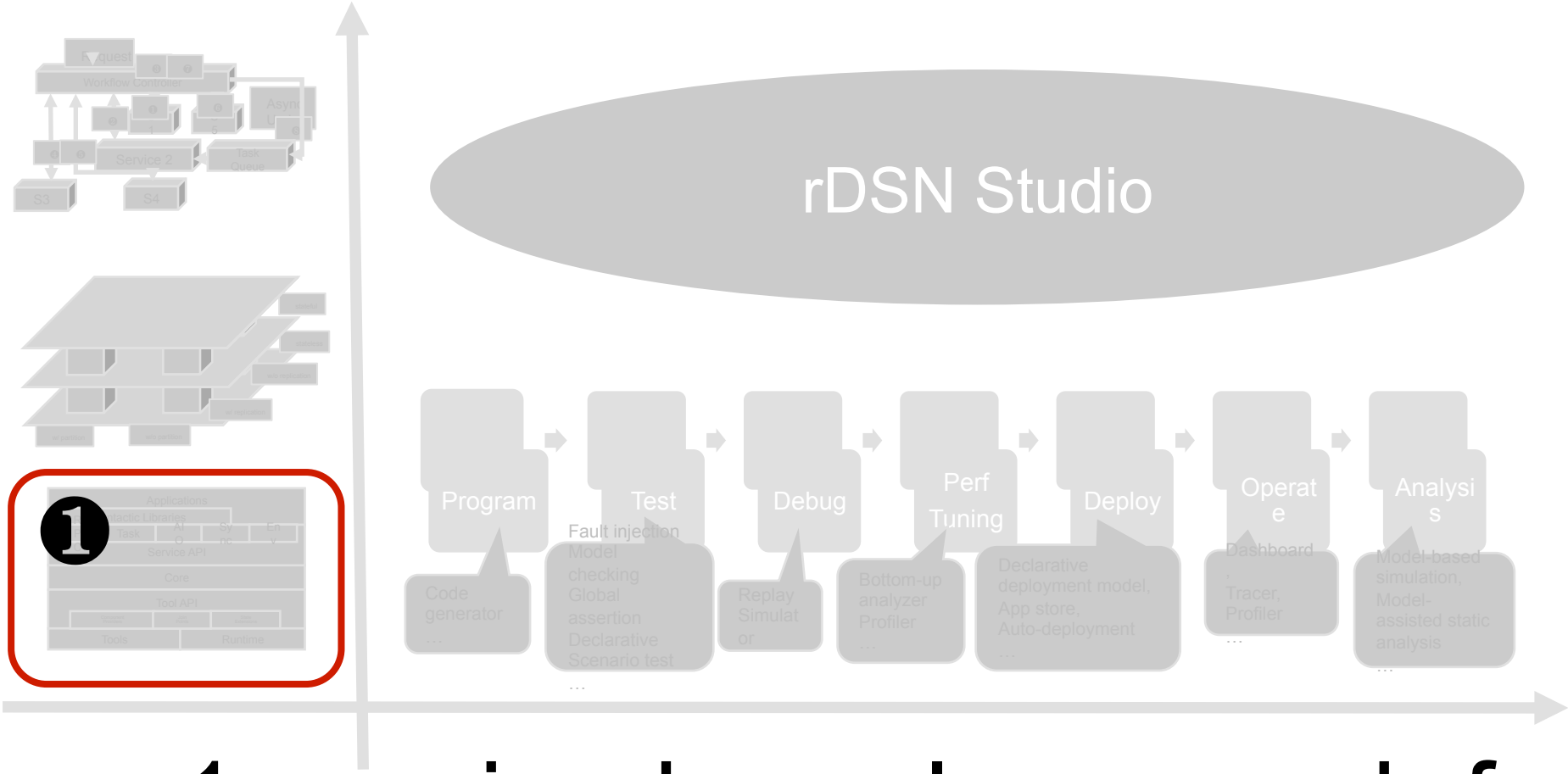


All together: (1). enhance service robustness from both runtime support and engineering process; (2). integrate all into a shared dev & op platform



rDSN Deep Dive


3
Request
Workflow Controller
S1
S5
Async Update
Service 2
Task Queue
S3
S4
2
1
Applications
Syntactic Libraries
Task
AIO
Env
Service API
Core
Tool API
Tools
Runtime
5
rDSN Studio
4
Program
Code generator
Test
Fault injection
Model checking
Global assertion
Declarative Scenario test
Debug
Replay Simulator
Perf Tuning
Bottom-up analyzer
Profiler
Deploy
Declarative deployment model,
App store,
Auto-deployment
Operate
Dashboard,
Tracer,
Profiler
Analysis
Model-based simulation,
Model-assisted static analysis



Layer 1: a microkernel approach for independent development & seamless integration of applications, frameworks, tools, and local libraries

Deployment model as with the microkernel architecture

- All module ***types*** (e.g., a service app or a tool) are materialized in dynamic linked libraries and registered into rDSN

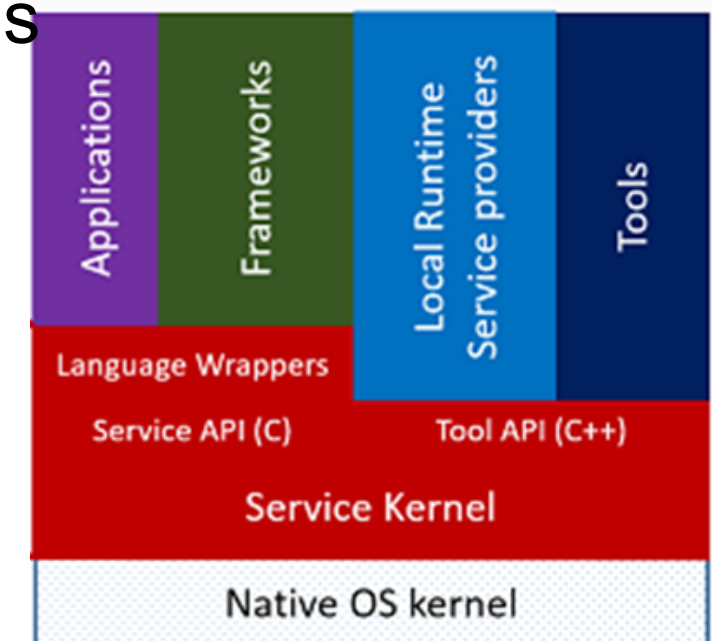
```
MODULE_INIT_BEGIN(tools_emulator)
    ::dsn::tools::register_component_provider< ::dsn::tools::sim_aio_provider>("dsn::tools::sim_aio_provider");
    ::dsn::tools::register_component_provider< ::dsn::tools::sim_network_provider>("dsn::tools::sim_network_provider");
    ::dsn::tools::register_component_provider< ::dsn::tools::sim_env_provider>("dsn::tools::sim_env_provider");
    ::dsn::tools::register_component_provider< ::dsn::tools::sim_task_queue>("dsn::tools::sim_task_queue");
    ::dsn::tools::register_component_provider< ::dsn::tools::sim_timer_service>("dsn::tools::sim_timer_service");
    ::dsn::tools::register_component_provider< ::dsn::tools::sim_semaphore_provider>("dsn::tools::sim_semaphore_provider");
    ::dsn::tools::register_component_provider< ::dsn::tools::sim_lock_provider>("dsn::tools::sim_lock_provider");
    ::dsn::tools::register_component_provider< ::dsn::tools::sim_lock_nr_provider>("dsn::tools::sim_lock_nr_provider");
    ::dsn::tools::register_component_provider< ::dsn::tools::sim_rwlock_nr_provider>("dsn::tools::sim_rwlock_nr_provider");

    dsn::tools::register_tool<dsn::tools::emulator>("emulator");
MODULE_INIT_END
```

- A loader (*dsn.svchost*) initiates the module instances

```
[modules]
dsn.app.simple_kv
dsn.tools.common
dsn.tools.emulator
```

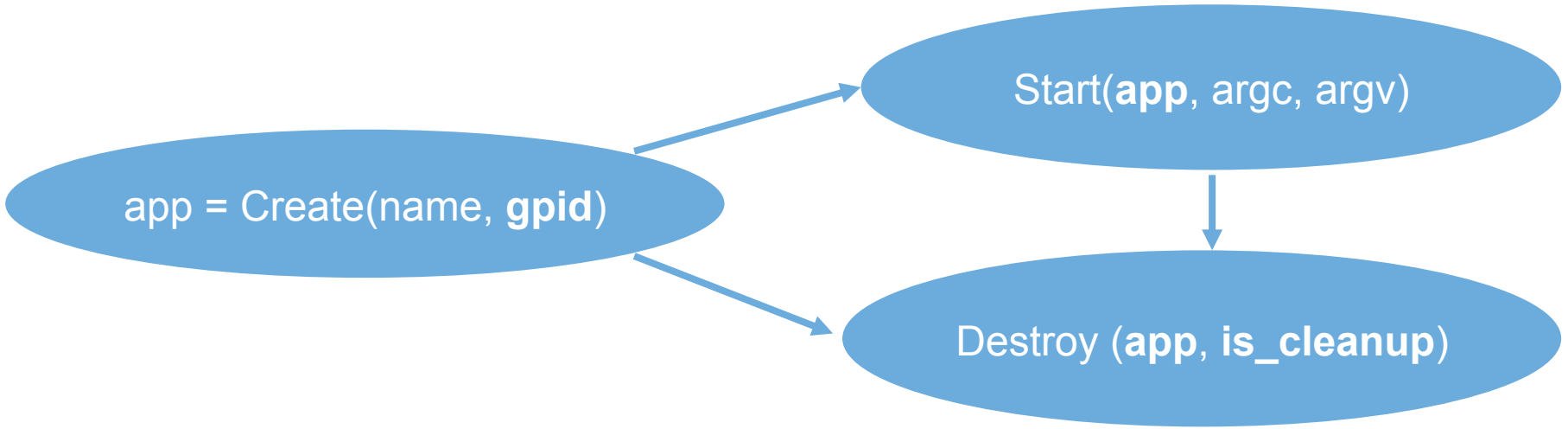
```
[apps.simple_kv]
type = simple_kv
arguments =
ports = 34801
run = true
count = 1
pools = THREAD_POOL_DEFAULT
```

```
[core]
;tool = nativerun
tool = emulator
```



Pluggable modules	Description	Release
dsn.core	rDSN service kernel	1.0.0
dsn.dist.service.stateless	scale-out and fail-over for stateless services (e.g., micro services)	1.0.0
dsn.dist.service.stateful.type1	scale-out, replicate, and fail-over for stateful services (e.g., storage)	1.0.0
dsn.dist.service.meta_server	membership, load balance, and machine pool management for the above service frameworks	1.0.0
dsn.dist.uri.resolver	a client-side helper module that resolves service URL to target machine	1.0.0
dsn.dist.traffic.router	fine-grain RPC request routing/splitting/forking to multiple services (e.g., A/B test)	todo
dsn.tools.common	deployment runtime (e.g., network, aio, lock, timer, perf counters, loggers) for both Windows and Linux; simple toollets, such as tracer, profiler, and fault-injector	1.0.0
dsn.tools.nfs	an implementation of remote file copy based on rpc and aio	1.0.0
dsn.tools.emulator	an emulation runtime for whole distributed system emulation with auto-test, replay, global state checking, etc.	1.0.0
dsn.tools.hpc	high performance counterparts for the modules as implemented in tools.common	todo
dsn.tools.explorer	extracts task-level dependencies automatically	1.0.0
dsn.tools.log.monitor	collect critical logs (e.g., log-level >= WARNING) in cluster	1.0.0
dsn.app.simple_kv	an example application module	1.0.0

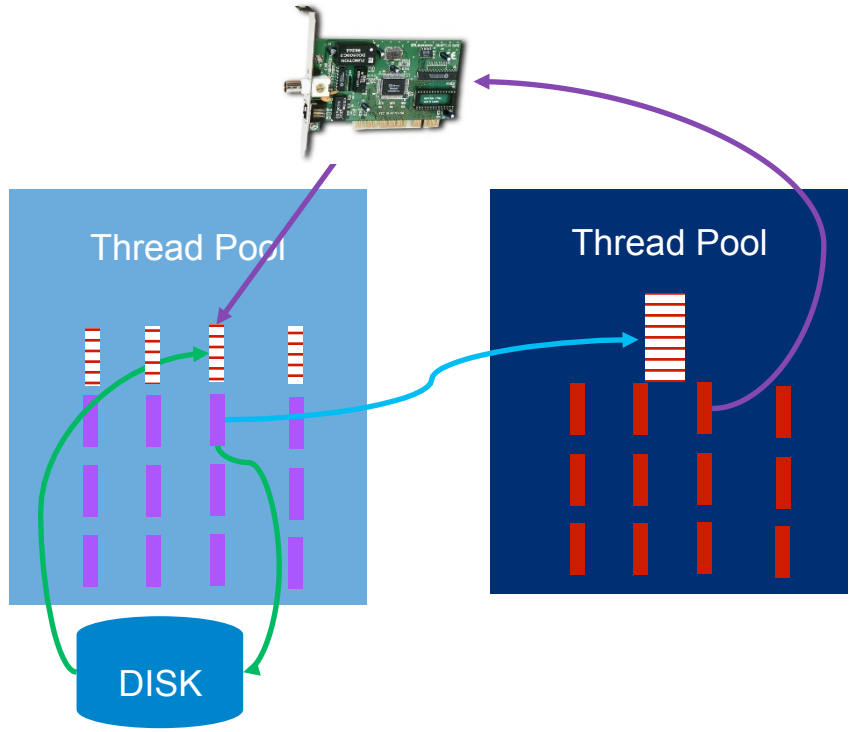
Application model (layer 1)




```
[apps.client]

type = client
arguments = localhost:34801
run = true
count = 1
pools = THREAD_POOL_DEFAULT
```

- *Start* is similar to traditional *main(argc, argv)*
- + service support: *gpid* (global service partition identity)
- + tool support
  - multiple application state allowed in the same process
  - application state can be cleaned-up if necessary (in *Destroy*)

Task-centric (event-driven) execution model with fine-grain resource configuration



Thread Pools

Task queue

Tasks
(Events)

Network Task

Local Task

Disk Task

Rethink the resource contention example at the beginning …

```
[task.RPC_PREPARE]
pool = THREAD_POOL_REPLICATION
is_trace = true
is_profile = false
; …
```

```
[thread_pool.THREAD_POOL_REPLICATION]
worker_count = 8
partitioned = true
priority = THREAD_PRIORITY_ABOVE_NORMAL
Affinity = 0x10
; …
```

Service API

Common Task Operations
Asynchronous Tasks and Timers
Remote Procedure Call (RPC)
File Operations
Environment
Thread Synchornization

Error Code
Command-Line Interface (cli)
Configuration Service
Logging Service
Checksum
Performance Counters
Memory Management

- Always trade-off between programming generality and agility
- Considerations
  - Target service applications (no UI)
  - Following common programming practice today
  - Be inclusive
    - API in C, wrapped in other languages for cross language integration
  - + native tooling support
    - **Non-deterministic behavior must go through this API for being monitored and manipulated**
    - With appropriate application level semantic

Example: a simple counter service

Client

```
// RPC_COUNTER_COUNTER_READ
template<typename TCallback>
::dsn::task_ptr read(
    const std::string& name,
    TCallback&& callback, // RPC_COUNTER_COUNTER_READ_ACK
    std::chrono::milliseconds timeout = std::chrono::milliseconds(0),
    int request_thread_hash = 0, // RPC_COUNTER_COUNTER_READ
    uint64_t r                                          ER_COUNTER_READ
    int reply_                                          _READ_ACK
    dsn::optio                                          dsn::none
    )
{
    return ::d
                                                        t32_t>& reply)
                                                        nted) " << std::endl;
    request_partition_hash,
    reply_thread_hash
    );
}
```

```
namespace cpp dsn.example

struct count_op
{
    1: string name;
    2: i32    operand;
}

service counter
{
    i32 add(1:count_op op);
    i32 read(1:string name);
}
```

Server

```
void open_service(dsn_gpid gpid)
{
    this->register_async_rpc_handler(RPC_COUNTER_COUNTER_ADD, "add", &counter_service::on_add, gpid);
    this->register_async_rpc_handler(RPC_COUNTER_COUNTER_READ, "read", &counter_service::on_read, gpid);
}
```

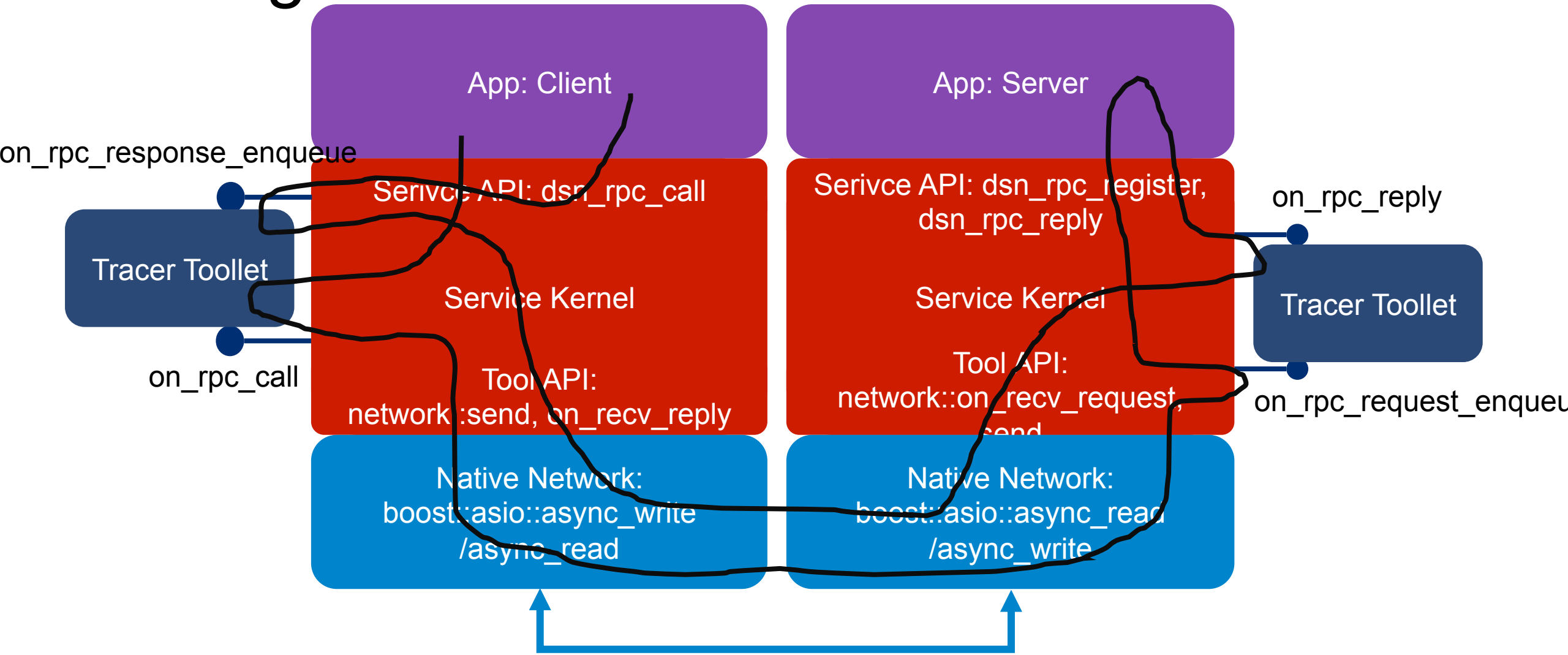
```
c:\Work\rDSN\tutorial>dsn.cg.bat counter.thrift cpp th
exec: C:\Work\rDSN\install\bin/Windows/thrift --gen rdsn -out th counter.thrift
exec: C:\Work\rDSN\install\bin/Windows/thrift --gen cpp:moveable_types -out th counter.thrift
generate 'th/CMakeLists.txt' successfully!
generate 'th/counter.app.example.h' successfully!
generate 'th/counter.check.h' successfully!
generate 'th/counter.client.h' successfully!
generate 'th/counter.client.perf.h' successfully!
generate 'th/counter.code.definition.h' successfully!
generate 'th/config.deploy.ini' successfully!
generate 'th/config.ini' successfully!
generate 'th/counter.local.client-server.ini' successfully!
generate 'th/counter.local.service.stateful.ini' successfully!
generate 'th/counter.main.cpp' successfully!
generate 'th/counter.server.h' successfully!
generate 'th/counter.types.h' successfully!
```

```
    int32_t resp;
    reply(resp);
}
```

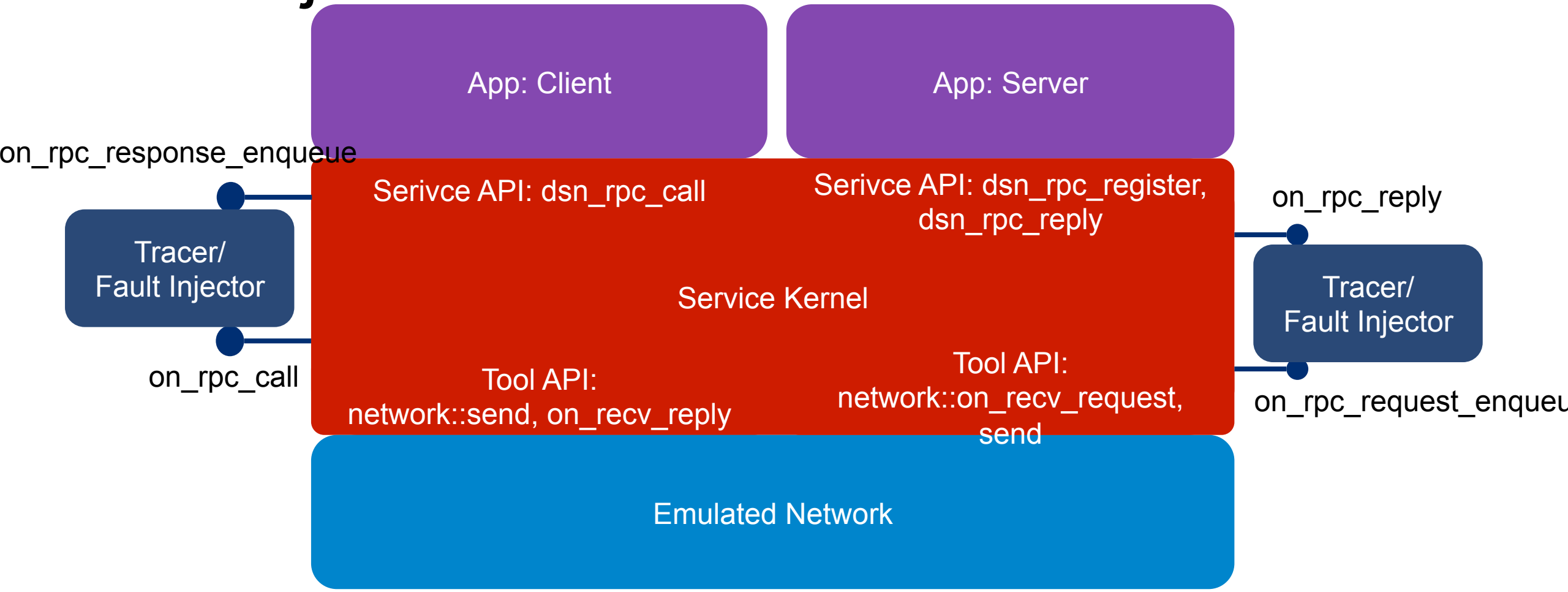
Tool abstractions and API

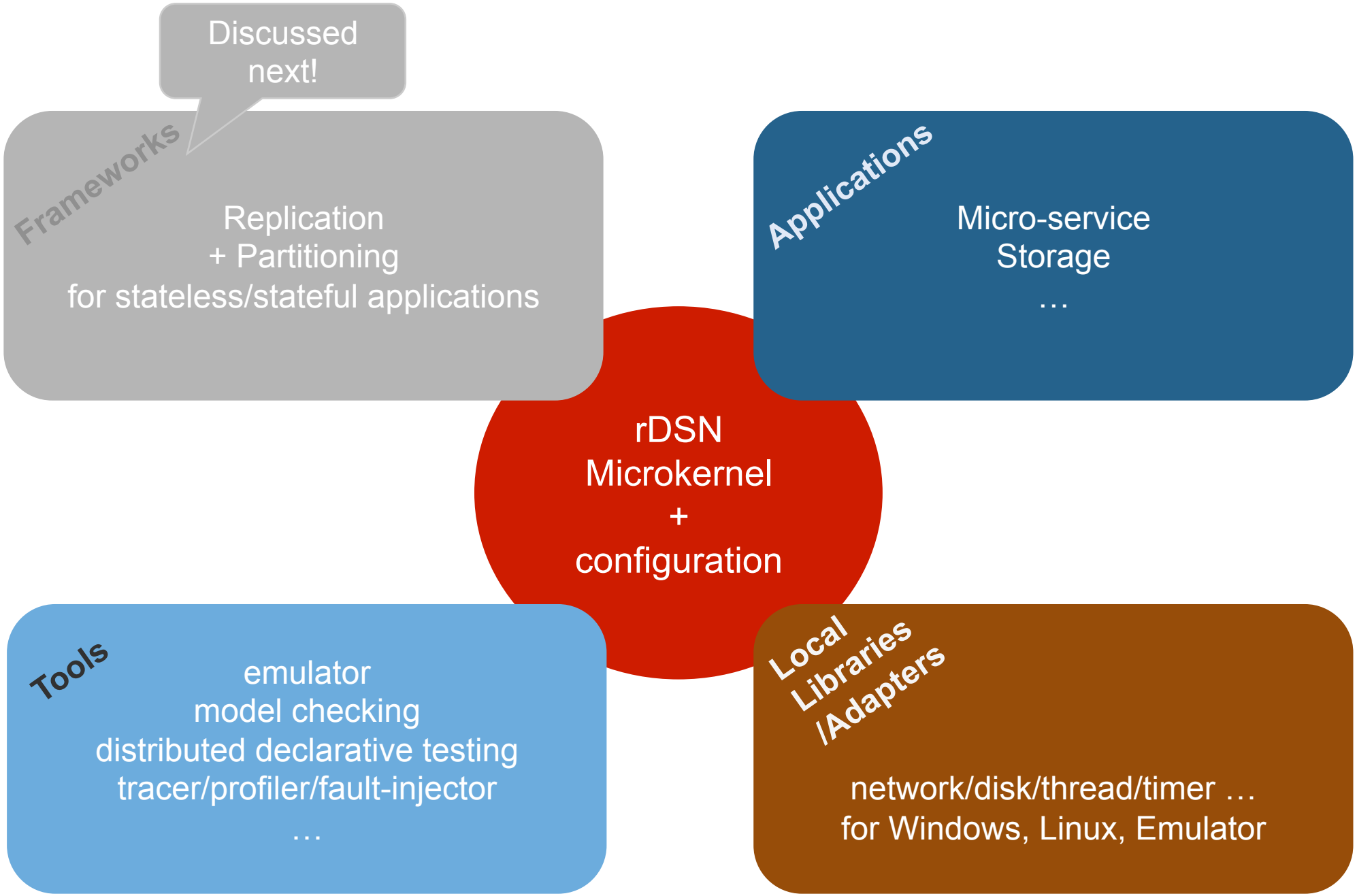
- Dedicated tool API for **tool & local library** development
- Three levels of tooling capability
  - Monitor
  - Tainting
  - Manipulation
- API: **complete interposition for reliably controlling all dependencies and non-determinisms in the system**
  - Component providers
    - network, disk, task_queue, lock, perf_counter, logger, …
  - Join points (hooks)
    - monitor and manipulate all task-task and task-env interactions
  - State extensions
    - task, message, ...
- Two kinds of tools
  - “tools”: only one tool exist in one process
  - “toollets”: multiple tools may co-exist

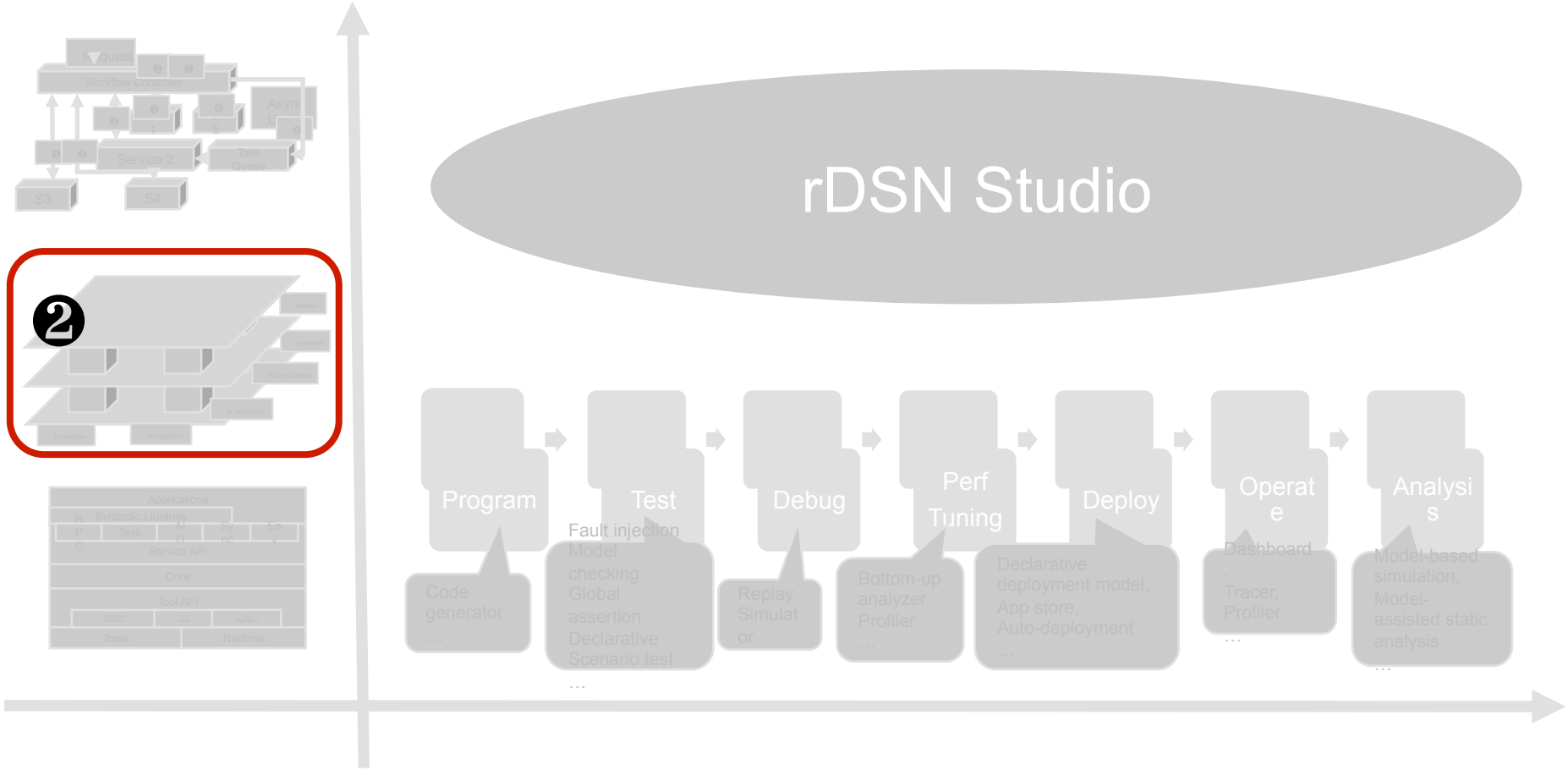
Example: network providers and RPC tracing



Or, we use emulated network, tracer, and fault injector

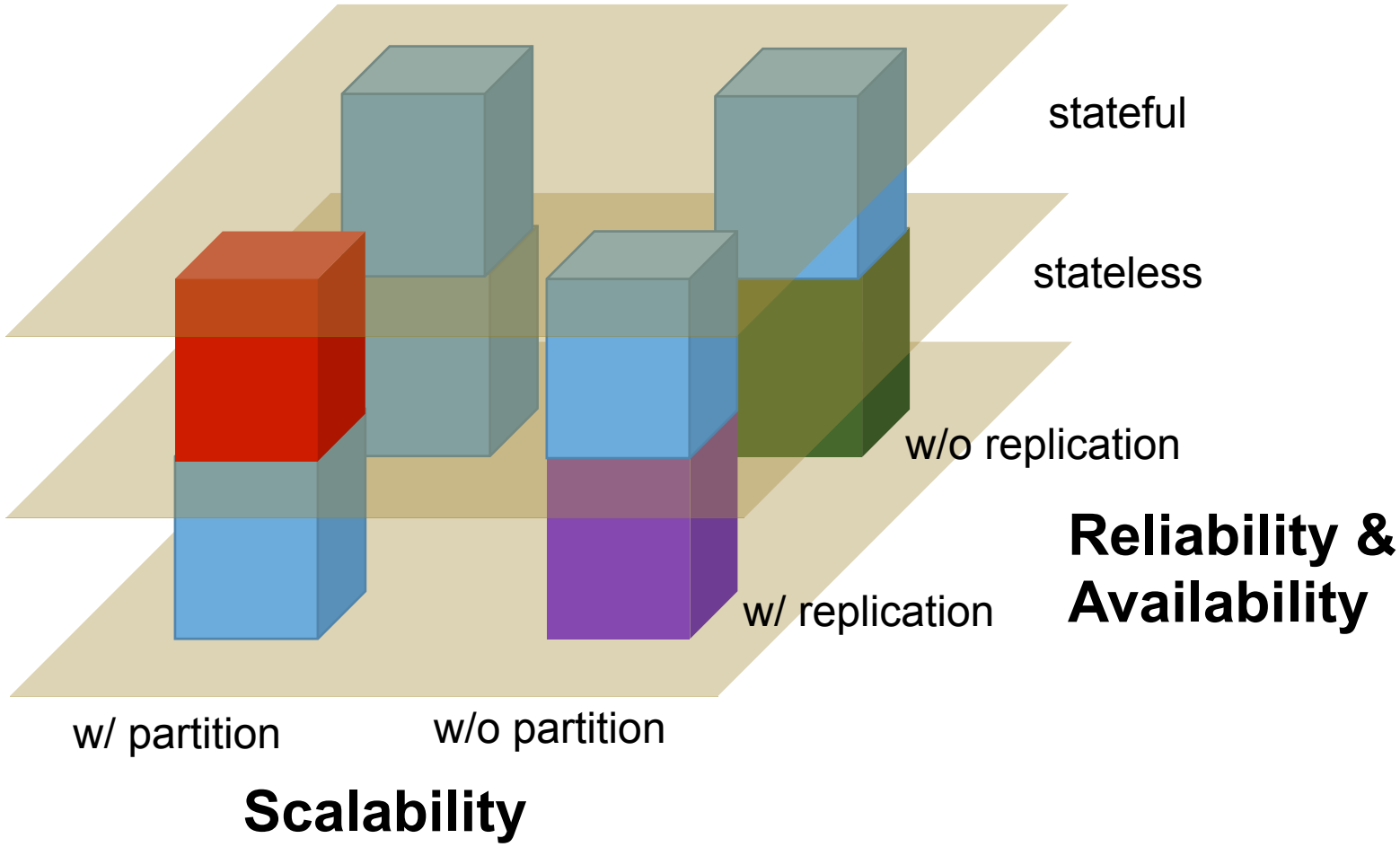



Discussed next!
Frameworks
Replication
+ Partitioning
for stateless/stateful applications
Applications
Micro-service
Storage
…
rDSN
Microkernel
+
configuration
Tools
emulator
model checking
distributed declarative testing
tracer/profiler/fault-injector
…
Local Libraries /Adapters
network/disk/thread/timer …
for Windows, Linux, Emulator

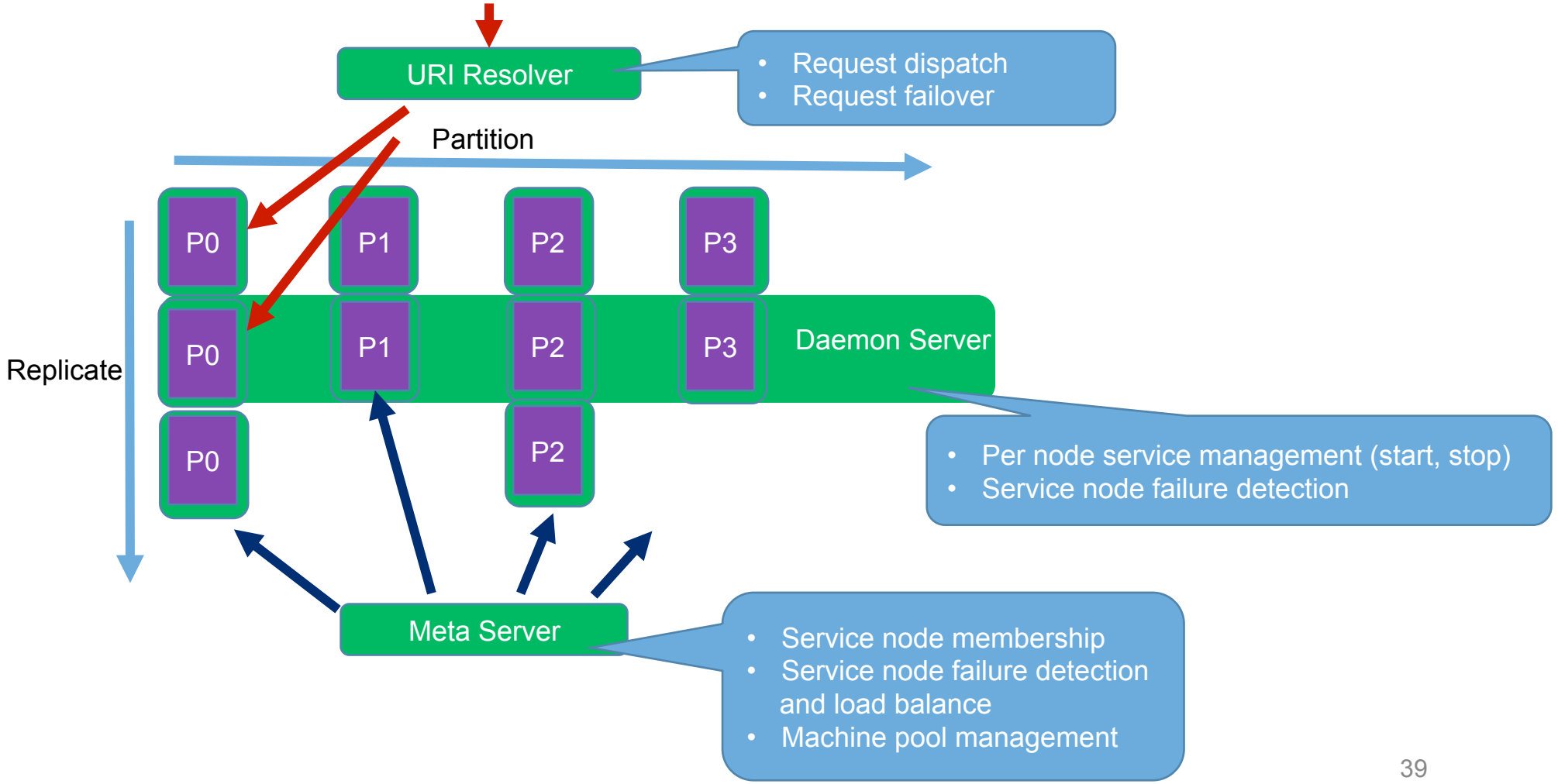


Layer 2: turn single-box components into scalable and reliable service

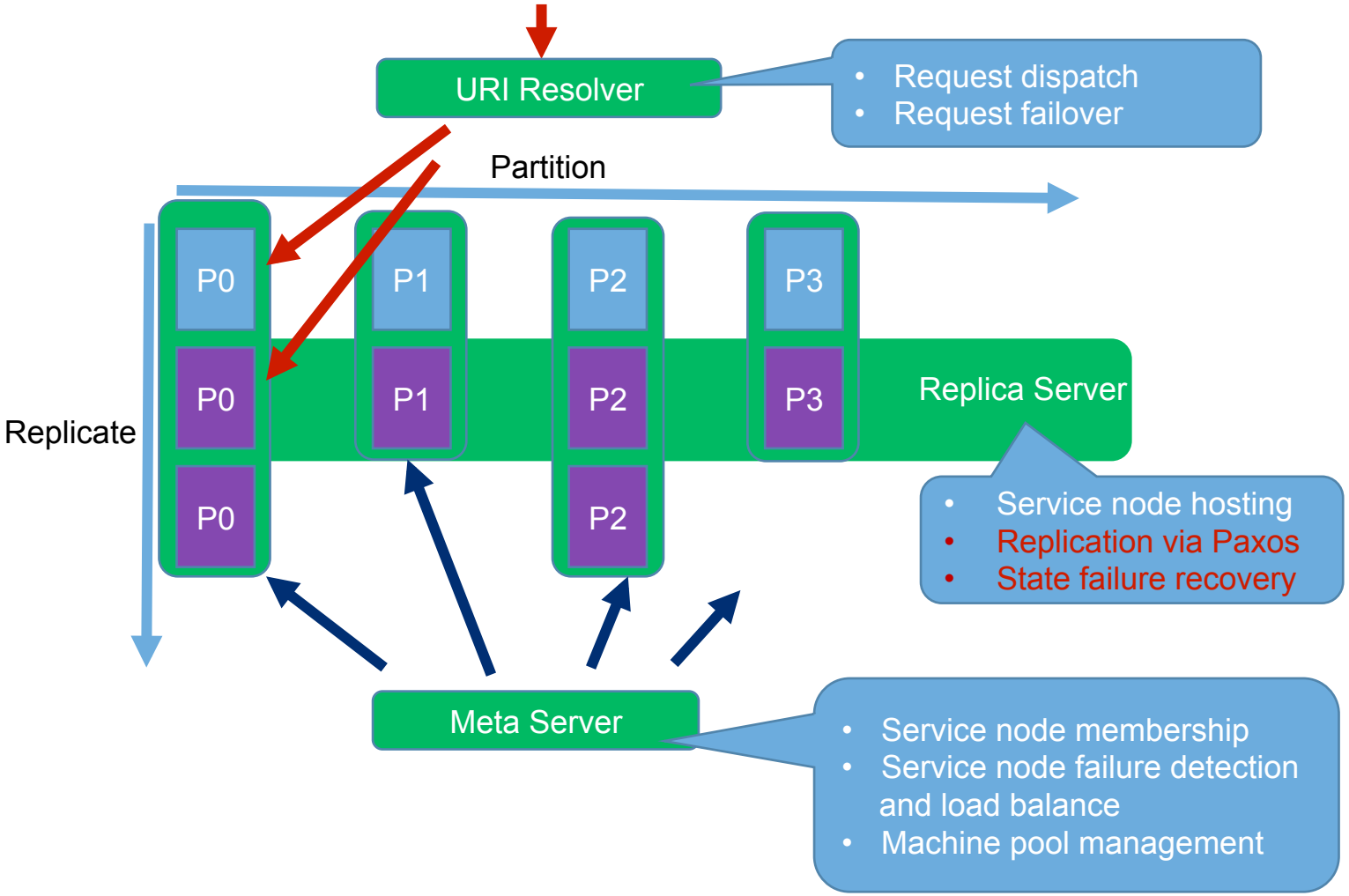
Service model for all rDSN applications



From single component to scalable and highly-available service (stateless) (: frameworks)



From single component to scalable and highly-available service (stateful) (: frameworks)



Application enhancement for being services

Client: use service URL as the target address

```
// RPC_COUNTER_COUNTER_READ
template<typename TCallback>
::dsn::task_ptr read(
    const std::string& name,
    TCallback&& callback, // RPC_COUNTER_COUNTER_READ_ACK
    std::chrono::milliseconds timeout = std::chrono::milliseconds(0),
    int request_thread_hash = 0, // RPC_COUNTER_COUNTER_READ
    uint64_t request_partition_hash = 0, // RPC_COUNTER_COUNTER_READ
    int reply_thread_hash = 0, // RPC_COUNTER_COUNTER_READ_ACK
    dsn::optional< ::dsn::rpc_address> server_addr = dsn::none
    )
{
    return ::dsn::rpc::call(
                server_addr.unwrap_or(_server),
                RPC_COUNTER_COUNTER_READ,
                name,
                this,
                std::forward<TCallback>(callback),
                timeout,
                request_thread_hash,
                request_partition_hash,
                reply_thread_hash
                );
}
```

Server: implement several helper functions (stateful service only)

```
virtual ::dsn::error_code sync_checkpoint(int64_t last_commit) override;

virtual int64_t get_last_checkpoint_decree() override { return last_durable_decree(); }

virtual ::dsn::error_code get_checkpoint(
    int64_t learn_start,
    int64_t local_commit,
    void*   learn_request,
    int     learn_request_size,
    app_learn_state& state
) override;

virtual ::dsn::error_code apply_checkpoint(
    dsn_chkpt_apply_mode mode,
    int64_t local_commit,
    const dsn_app_learn_state& state
) override;
```

Adoption example

- Rocksdb + dsn.dist.service.stateful in Xiaomi for replacing the original HBase/Redist clusters

- For HBase
  - No GC problems which leads to latency spikes
  - Faster failure recovery due to replicated application state (instead of only on-disk state)
- For redis
  - Better consistency guarantee and reconfiguration support

Framework development

- Frameworks are special apps that can host other apps
- Frameworks get access to application models through their *type*
  - *dsn_app_get_callbacks*
- Application messages are sent to frameworks first
  - Frameworks implement *on_rpc_request* to receive app messages
  - Frameworks send requests to apps through *dsn_hosted_app_commit_rpc_request*

Integrated into Xiaomi cluster managements

- Xiaomi has its own way to (so is Microsoft Bing)
  - Write and manage logs
  - Performance counter implementation and monitoring

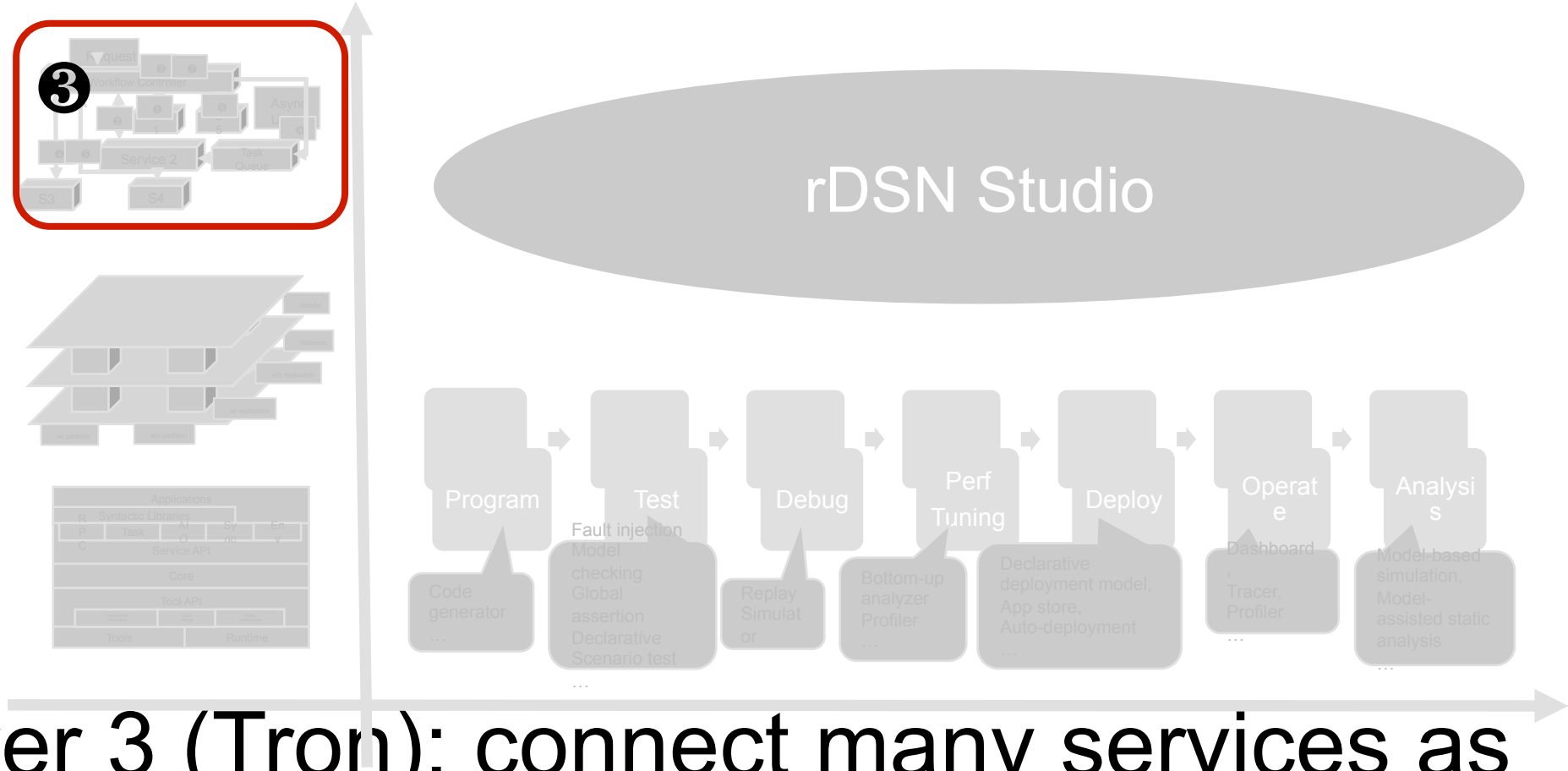
- Wrap the following components and plug into rDSN
  - Log providers – we can immediately use many old log collection and search tools
  - Performance counters – we can immediately reuse the dashboard previously designed for HBase

Integration with legacy Java programs

- Use Thrift generated java client

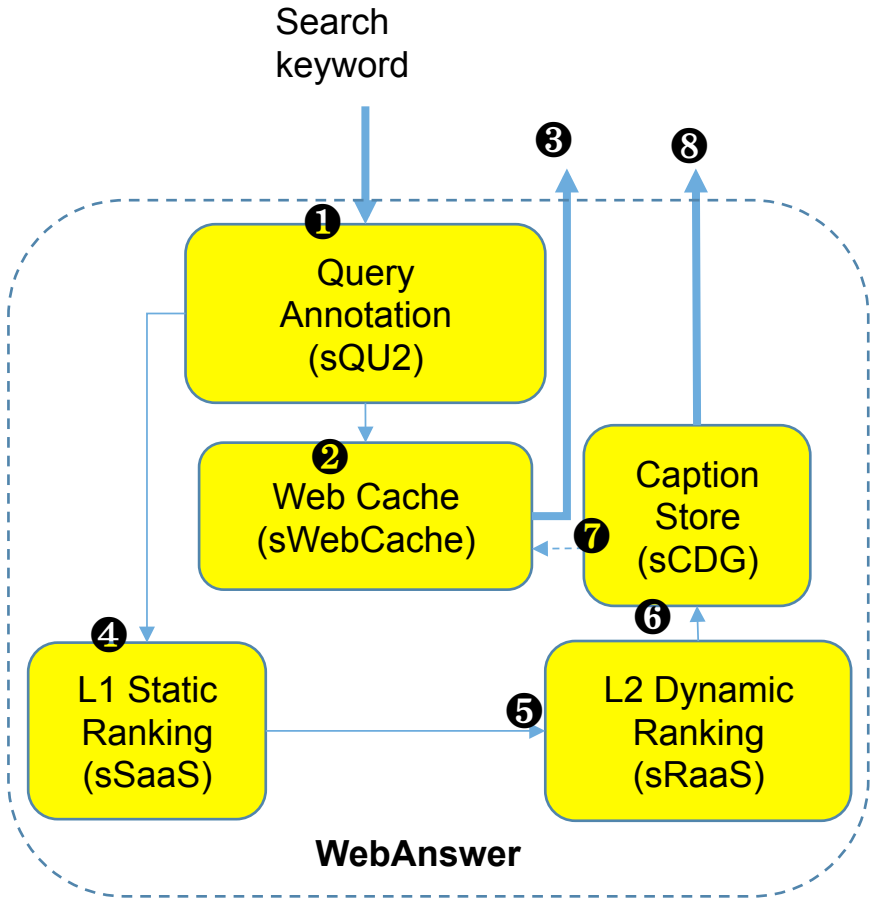
- Problem
  - Thrift header is different from rDSN
  - Thrift data encoding is different from that in rDSN

- Solution
  - Plugin customized message header parser (thrift_message_parser)
  - rDSN allows multiple payload encoding/decoding protocols simultaneously



Layer 3 (Tron): connect many services as flows to serve end-to-end user requests (in progress)

Language: declarative service composition



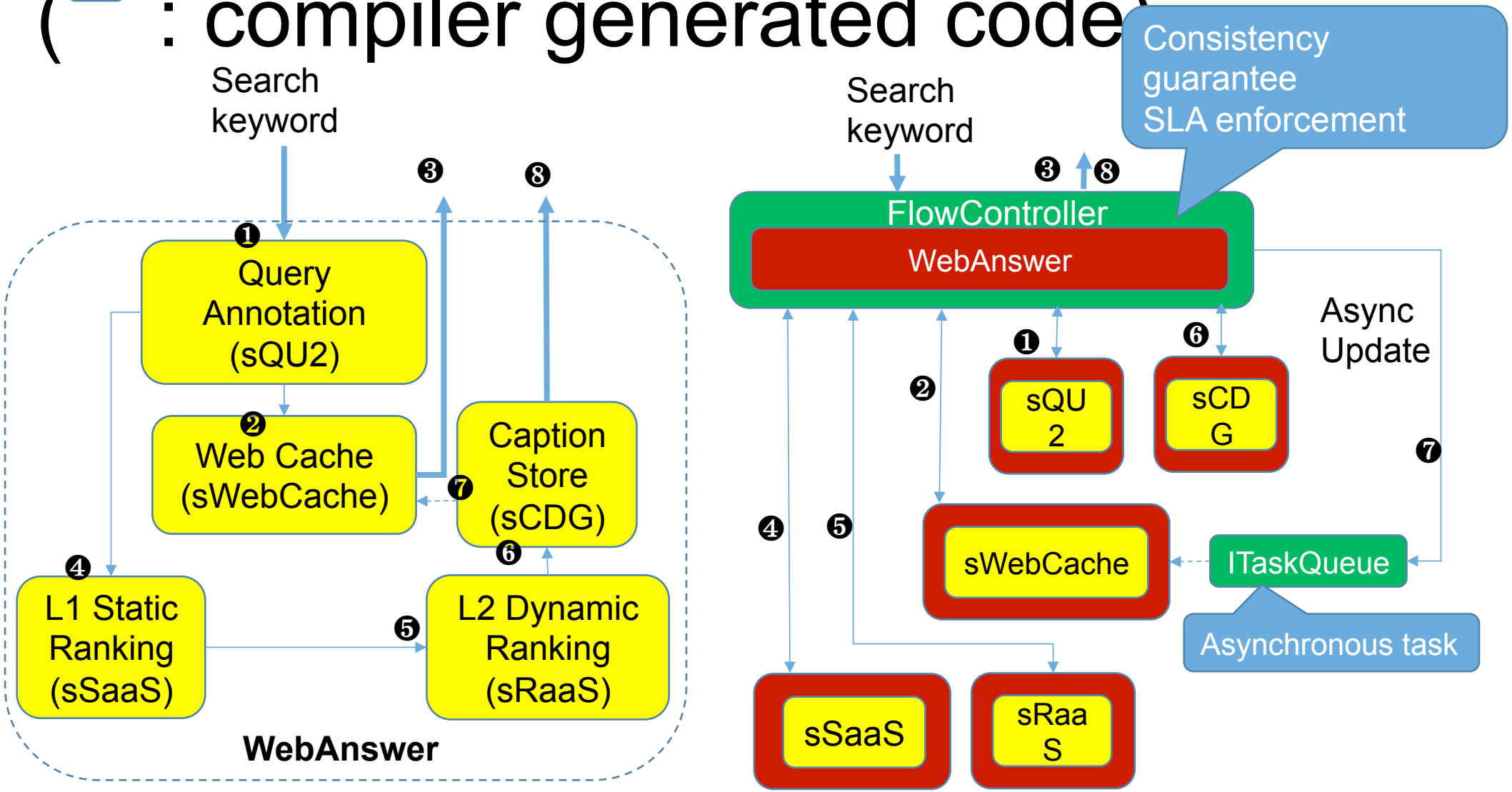

```
static IsCache_Svc sIsCache = new IsCache_Svc("IsCache")
public static ISymbol<QueryResult> WebAnswer(this ISymbol<StringQuery> iq)
{
    var aq = iq.Call(q => sQU2.OnQueryAnnotation(q));❶// query annotation

    return aq
        .Call(q => sWebCache.Get(q))❷// check for web cache, using augmente
        .IfThenElse(
            r => r.Results.Count > 0,
            hitPart => hitPart,❸
            missPart2 => missPart2
                .Call(q => sSaaS.OnL1Selection(q.Query))❹
                .Scatter(rr => rr.Results)
                .Top(s => s.StaticRank, 1000, false)
                .Call(r => new PerDocRank { Id = r.Pos.DocIdentity,
                                            Rank = sRaaS.OnL2Rank(r) })❺
                .Top(r => r.Rank.Value, 100, false)
                .Call(r => sCDG.Get(r.Id))❻
                .Gather(cs => new QueryResult
                {
                    Query = aq.AsValue(),
                    Results = cs.ToList()
                })❽
                .AsyncCall(ar => sWebCache.Put(ar))❼
            );
}
```

Why?

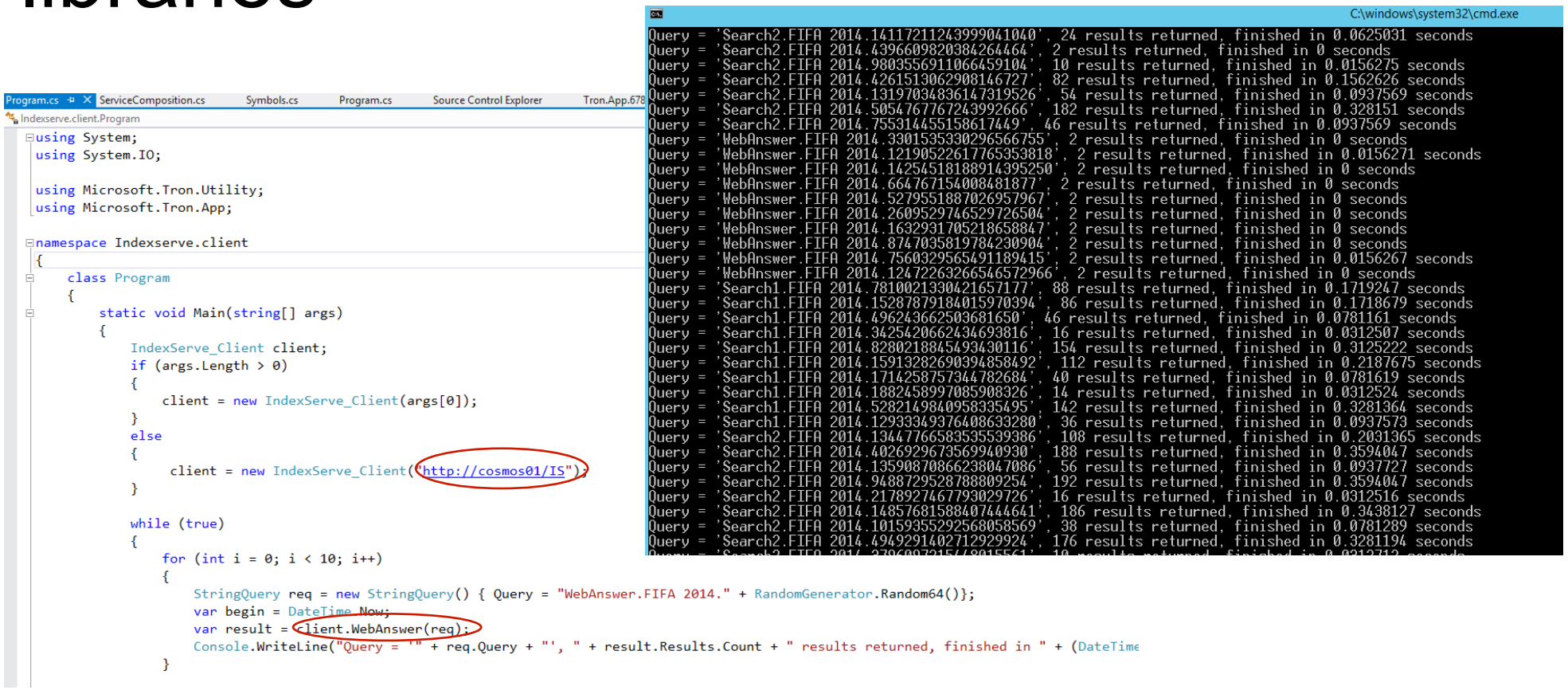
- Explicit cross-service dependency
- Agile programming for new services or service changes
- Automated global coordination
  - Latency optimization
    - Parallel service access
    - Delayed computation (when response is not needed for client response)
    - …
  - Consistency
    - E.g., atomicity
  - Testing, debugging
    - E.g., Service level tracing, profiling
- A lot can be done during service flow compilation and code generation for better robustness and performance!

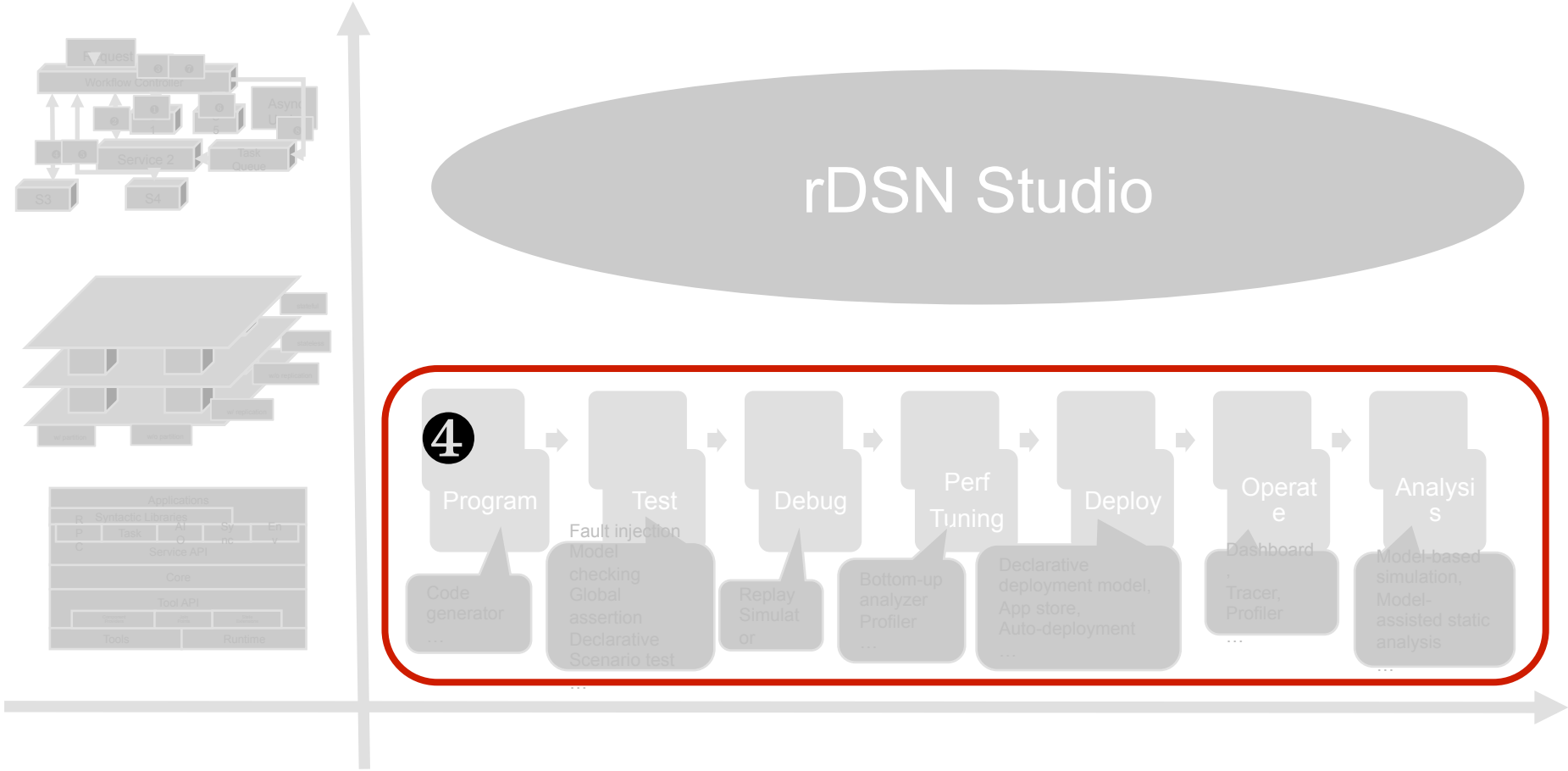
Deploy composed service as layer 2 services

(: compiler generated code)



Enjoy with auto-generated client libraries





Enhanced engineering process

Program

Code generator
…

Test

Fault injection
Model checking
Global assertion
Declarative Scenario test
…

Debug

Replay Simulator

Perf Tuning

Bottom-up analyzer
Profiler
…

Deploy

Declarative deployment model,
App store,
Auto-deployment
…

Operate

Dashboard,
Tracer,
Profiler
…

Analysis

Model-based simulation,
Model-assisted static analysis
…

Example: model checking (test) (in dsn.tools.emulator)

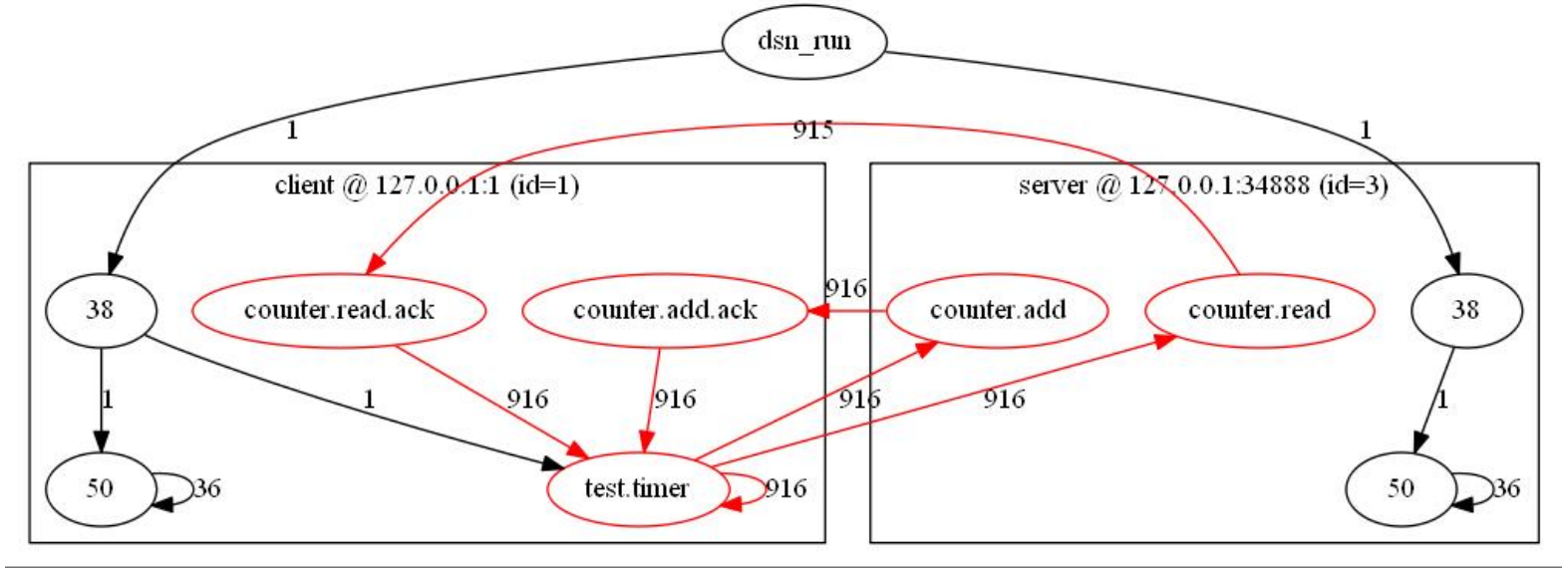
- Observation
  - Test coverage is always low compared to all the possible scenarios in reality
  - Difficult to test certain scenarios even if we know how they may happen
- Explore all possible scheduling and failure combinations
  - Thread interleaving
  - Message reordering, lost, delay
  - Disk slow, failure
- Due to the large state space, randomly select the possible choice at each join points
  - e.g., on_rpc_request_enqueue, on_aio_completed, …
- Effectively expose bugs that are difficult to found before

Example: replay (debug) (in dsn.tools.emulator)

- Observation: some subtle bugs are difficult to be reproduced, leading root cause analysis almost infeasible
- The random choices made in model checker are decided by a random seed
- Using the same seed leads to the same sequence of decisions (e.g., order, failure, …), therefore reproduces the same bug

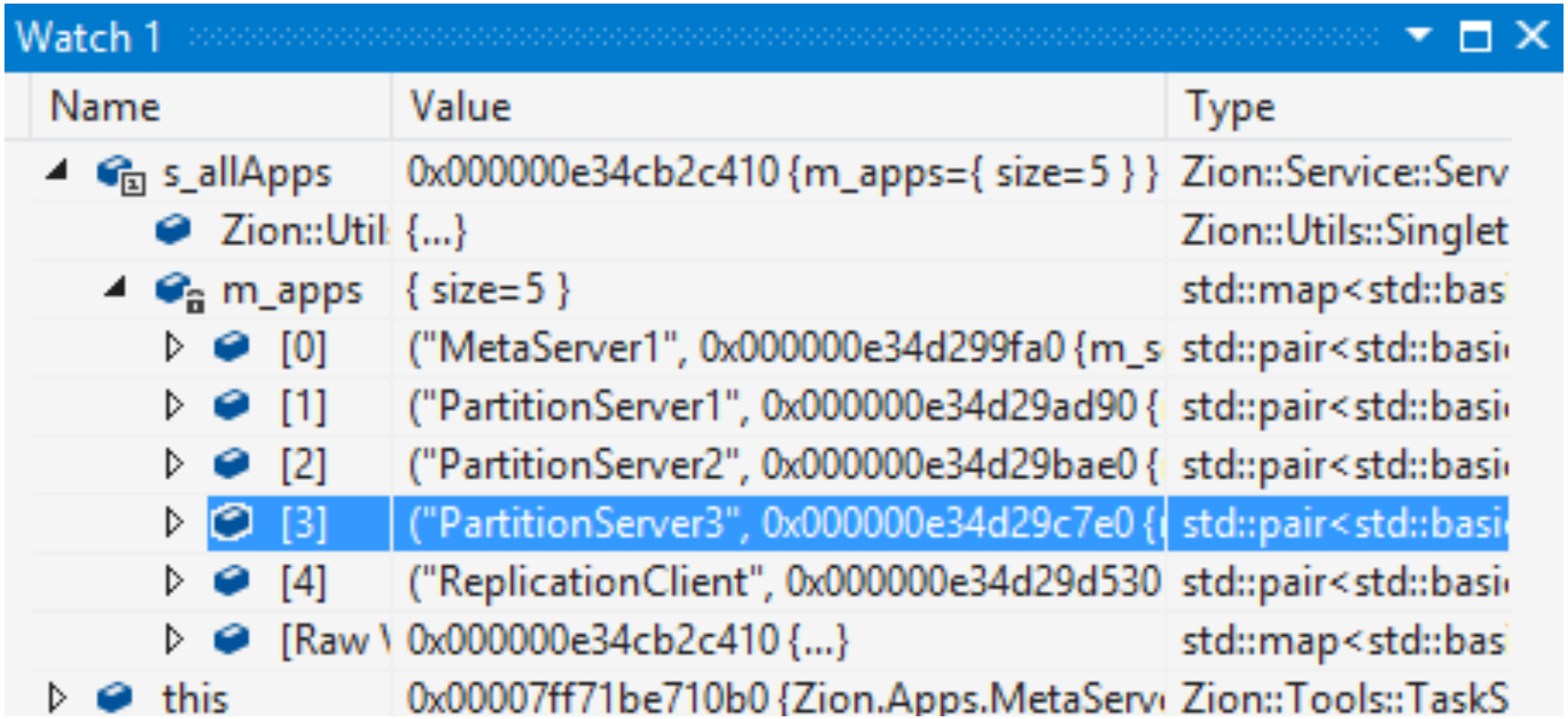
Example: task explorer (dsn.tools.explorer) (debug)

- Observation: when the system gets large, it is difficult for developers to figure out how the requests are processed in the system across threads and machines
- This tool automatically extracts such information and visualizes as task dependency graphs



Example: global state inspection (debug)
(in dsn.tools.emulator)

Observation: it is annoying when unwanted timeout happens after pausing in debuggers



Example: progressive system complexity (debug and performance tuning) (dsn.core + dsn.tools.common)

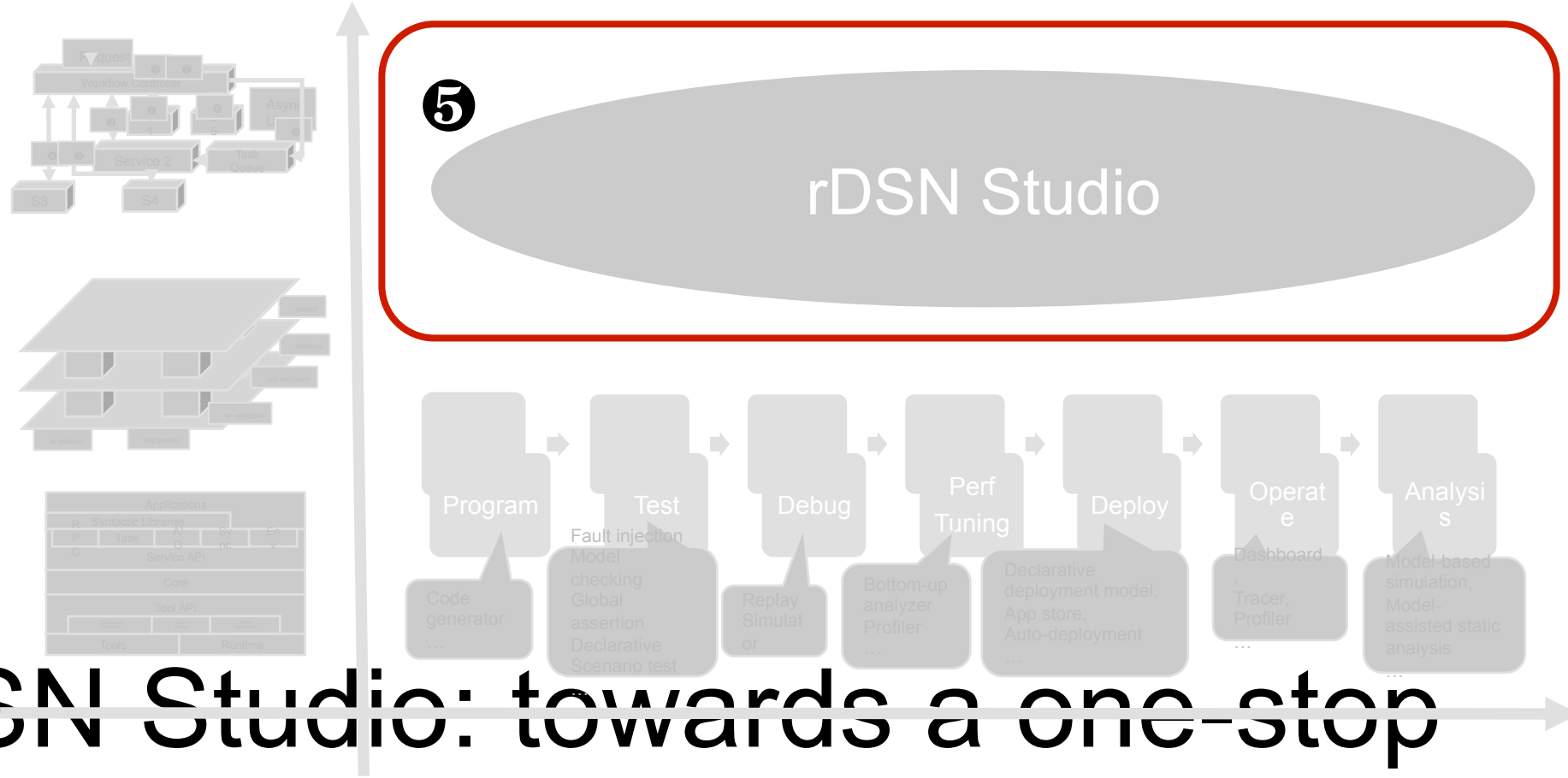
- Observation: it is difficult for developers to reasoning about big changes (which lead to either correctness or performance issues)

- Debug correctness and performance issues with controlled and gradually changed environments
  - Thread number
  - Network failure
  - Disk failure
  - Disk performance (delay, …)
  - Network performance (delay, …)
  - Node number
  - …

Example: declarative distributed testing (test, not generalized yet)

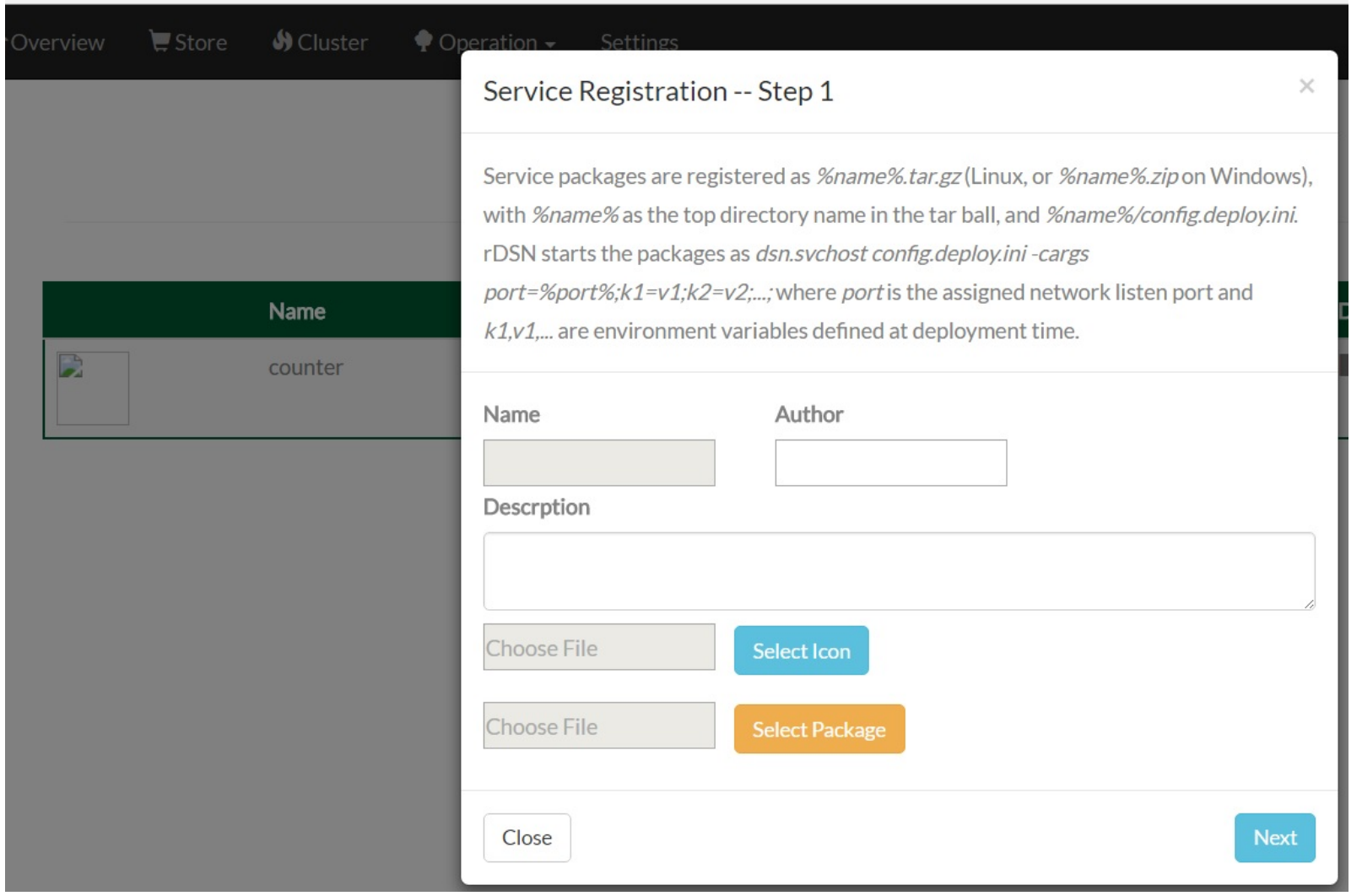
```
55 client:end_read:id=10,err=err_ok,resp=v10
56
57 # remove secondary r2
58 client:replica_config:receiver=r1,type=downgrade_to_inactive,node=r2
59 config:{3,r1,[]}
60 state:{{r1,pri,3,10}}
61
62 # add secondary r3 (new replica)
63 # will trigger learning in on_learn():
64 #   _prepare_list.count() {=10} > 0
65 #   learn_start_decree {=1} == _prepare_list->min_decree() {=1}
66 #   to-be-learn state is covered by prepare list, learn by CACHE, learn_mutation_count = 10
67 #
68 #                       (1)                 (10)
69 #   prepare_list :       |-------------------|
70 #   learn        :       |-->
71 #                       (1)
72 #
73 client:replica_config:receiver=r1,type=add_secondary,node=r3
74 config:{4,r1,[r3]}
75 state:{{r1,pri,4,10},{r3,sec,4,10}}
```

All are done by the tool modules, without changing any code of the upper applications and/or frameworks!

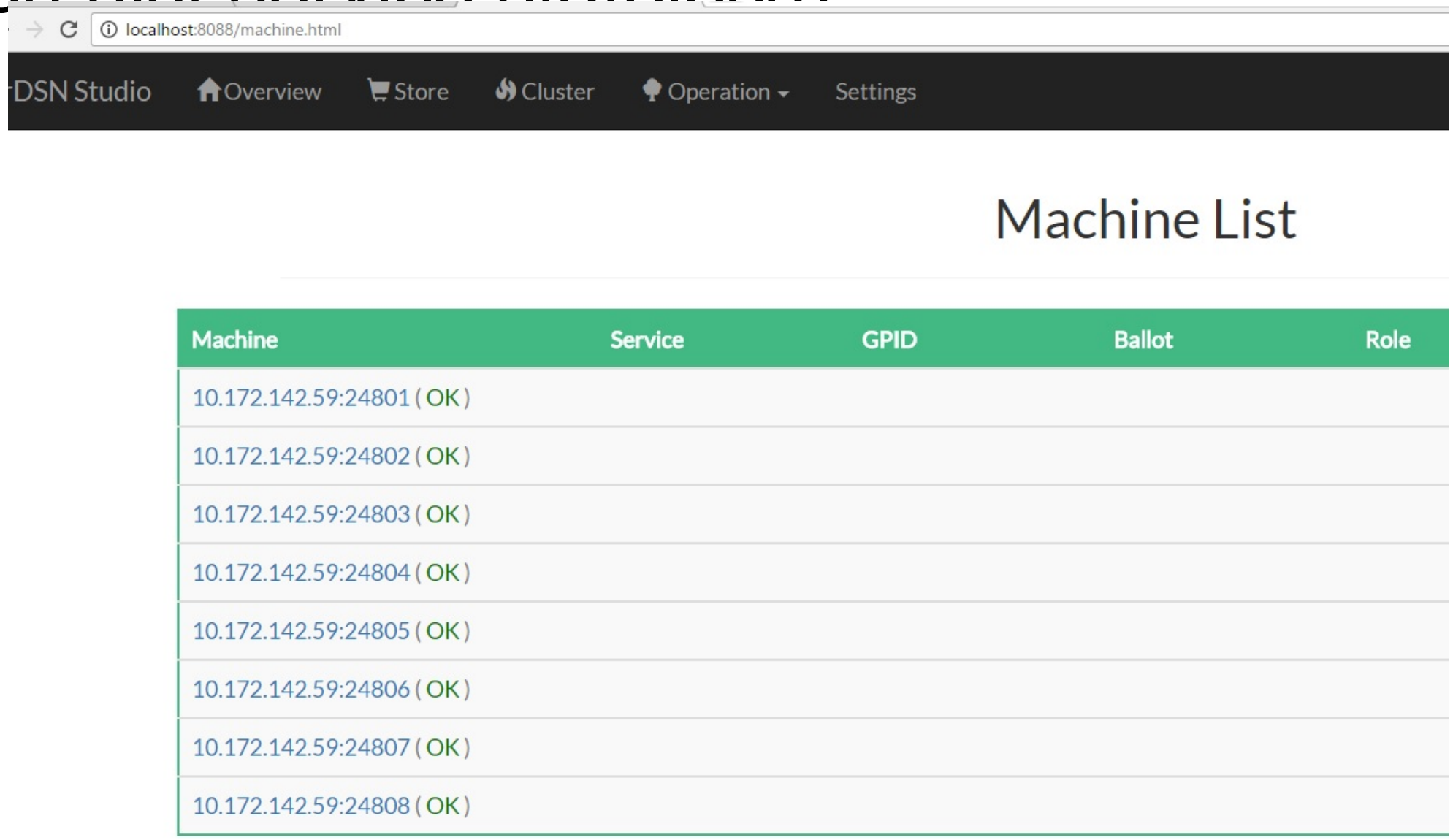


rDSN Studio: towards a one-stop IDOE for service development, sharing, discovery, deployment, and operation

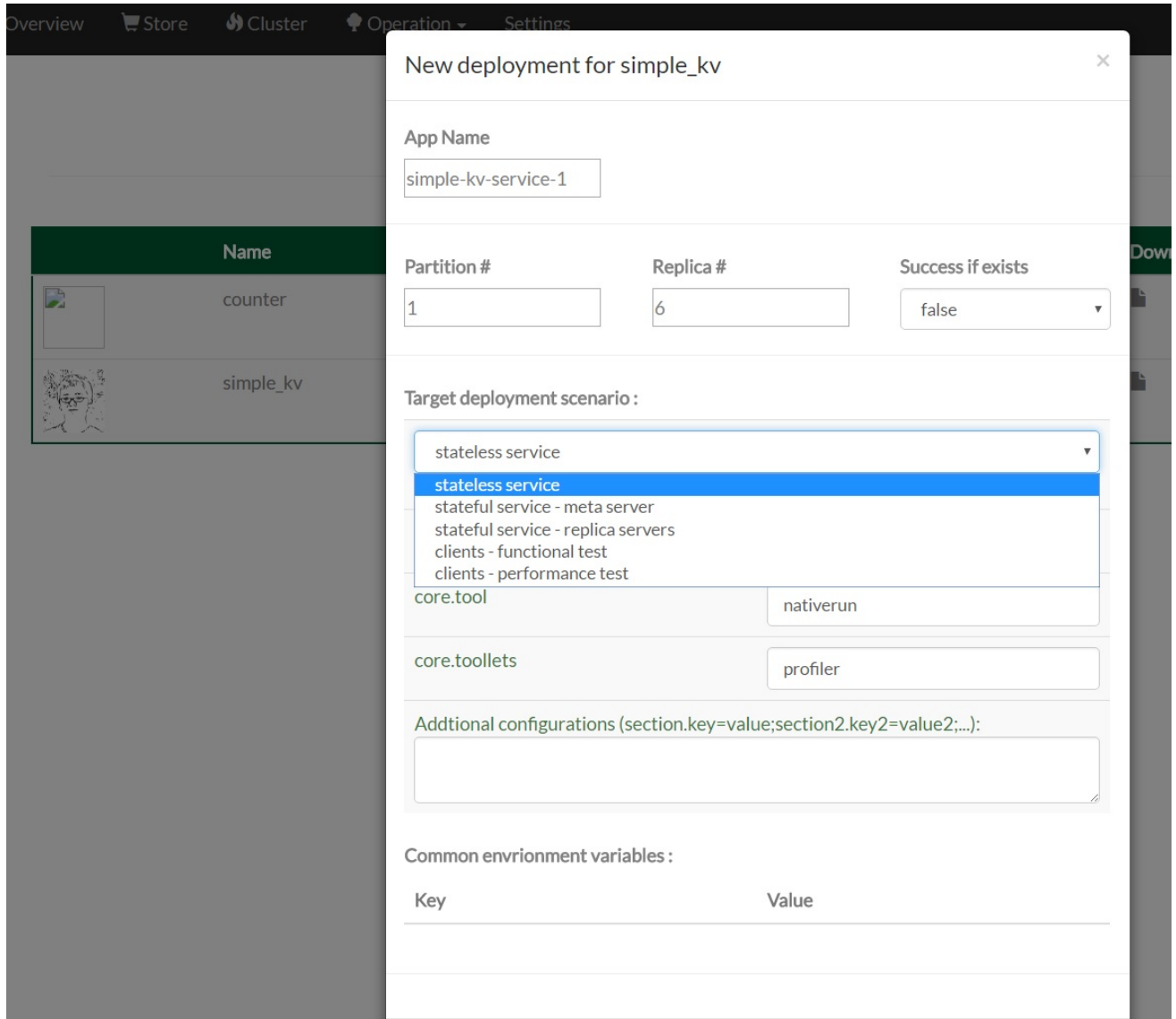
Service registration



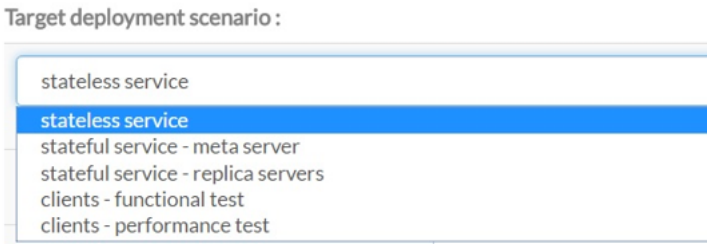
Cluster management
(dsn dist service stateless)



Service deployment

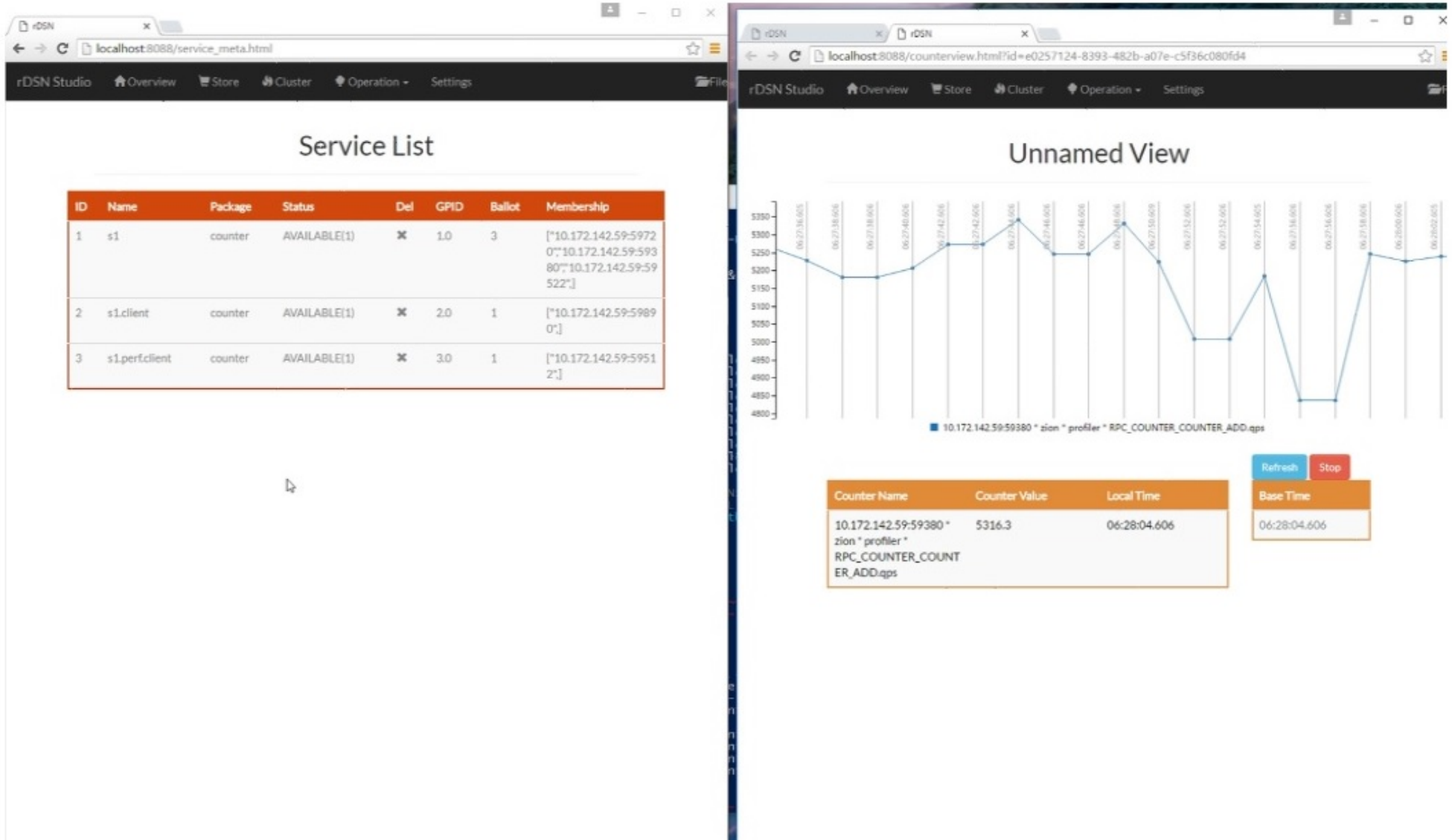


Configurations as scenarios

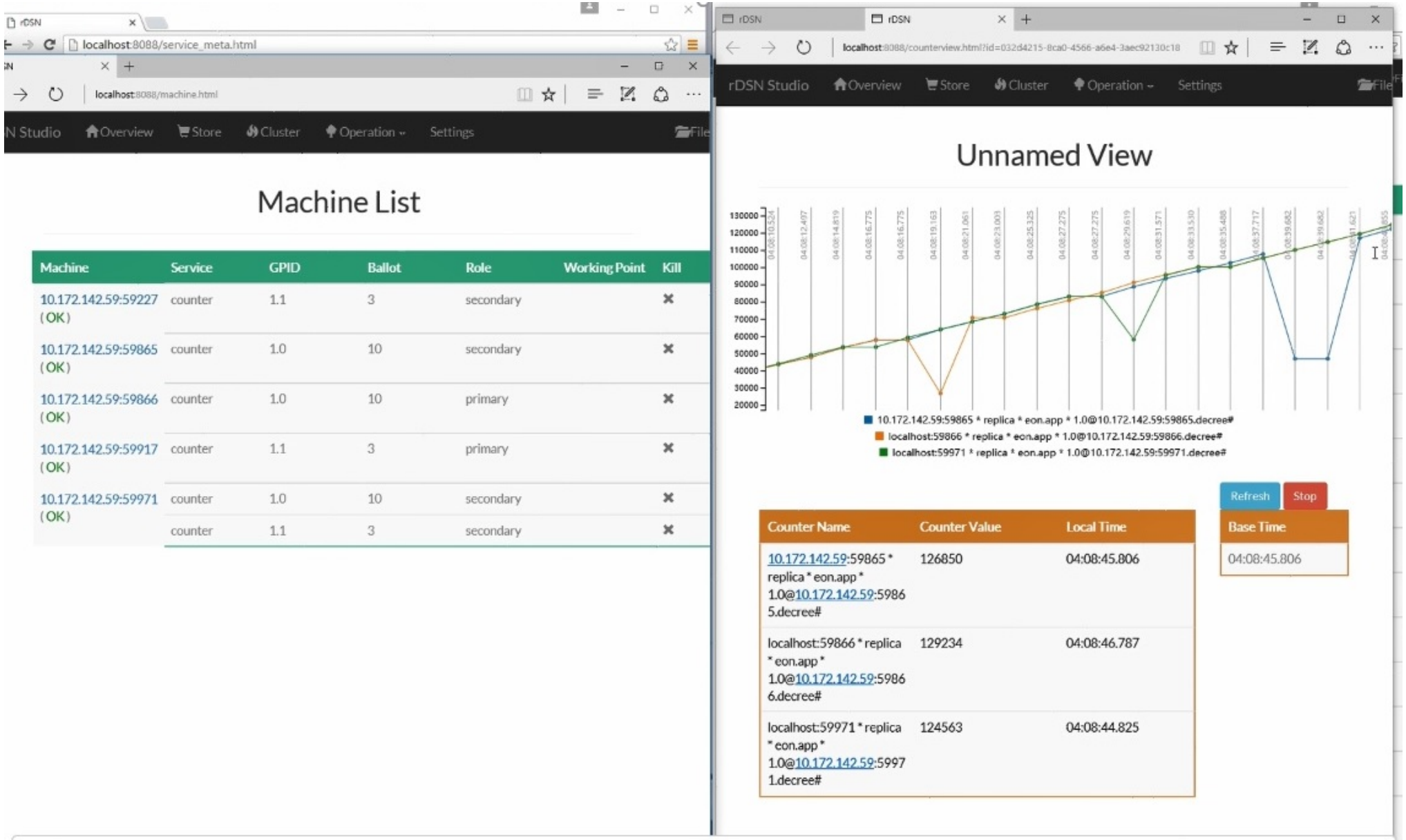



```
{
    "title" : "stateless service",
    "dsptr" : "serve as a stateless service",
    "config" : "config.deploy.ini",
    "overwrites" : {
        "apps.server.run" : "true",
        "core.tool" : "nativerun",
        "core.toollets" : "profiler",
        }
},
{
    "title" : "stateful service - meta server",
    "dsptr" : "start a meta server for managing the membership, load balancing etc. for a stateful service",
    "config" : "config.deploy.ini",
    "overwrites" : {
        "apps.meta.run" : "true",
        "core.tool" : "nativerun",
        "core.toollets" : "profiler,tracer",
        "replication.app.app_name" : "{{st}}",
        "replication.app.partition_count" : 4,
        "replication.app.max_replica_count" : 3,
        "meta_server.server_list" : "{{meta}}",
        "meta_server.min_live_node_count_for_unfreeze" : 1,
        }
},
{
    "title" : "stateful service - replica servers",
    "dsptr" : "start replica server(s) for hosting the service state and serving client requests",
    "config" : "config.deploy.ini",
    "overwrites" : {
        "apps.replica.run" : "true",
        "core.tool" : "nativerun",
        "core.toollets" : "profiler",
        "meta_server.server_list" : "{{meta}}",
        "uri-resolver.dsn://mycluster.arguments" : "{{meta}}",
    }
```

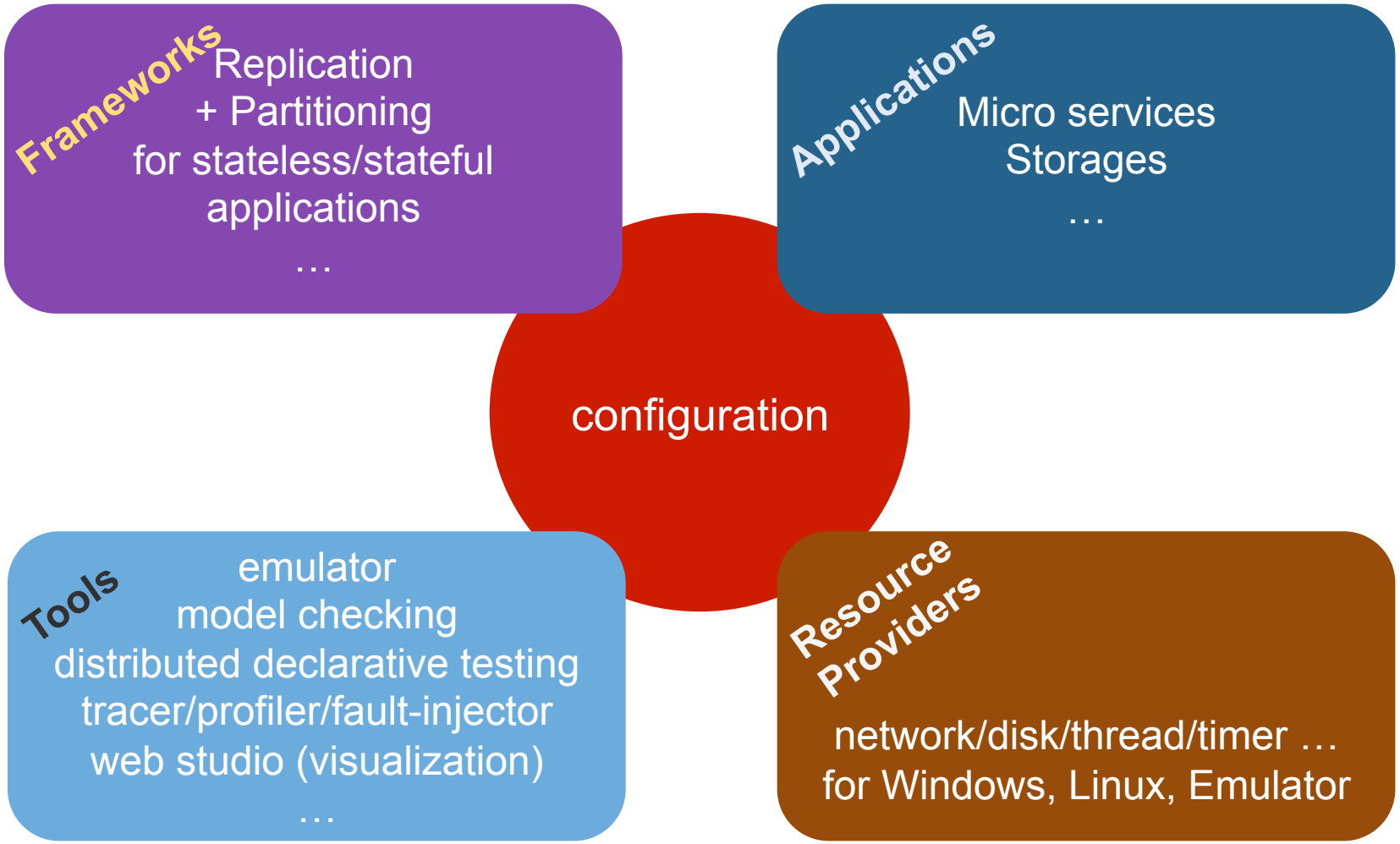
Service operation (stateless)



Service operation (stateful)



Share and Benefit



Key take away

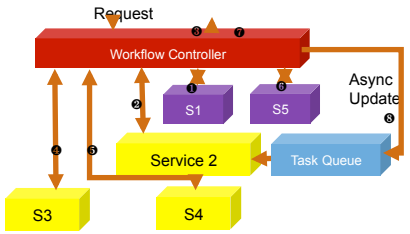
What is rDSN?

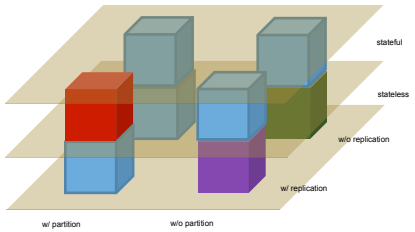
- Robustness is far from being perfect in current systems due to
  - Implicit contract
  - Lack of Systems Thinking
  - After-thought tooling support

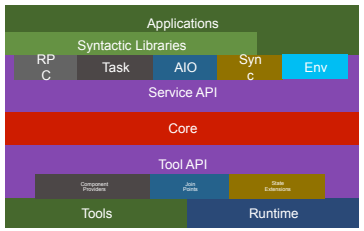
- rDSN advocates
  - Explicit task and service level dependency, non-determinism, and interference
  - Flexible configurations that break the dilemma between local decisions and global coordination
  - Native tooling support

Effort in (1) both runtime and engineering process; (2) a coherent way that they can be seamlessly integrated

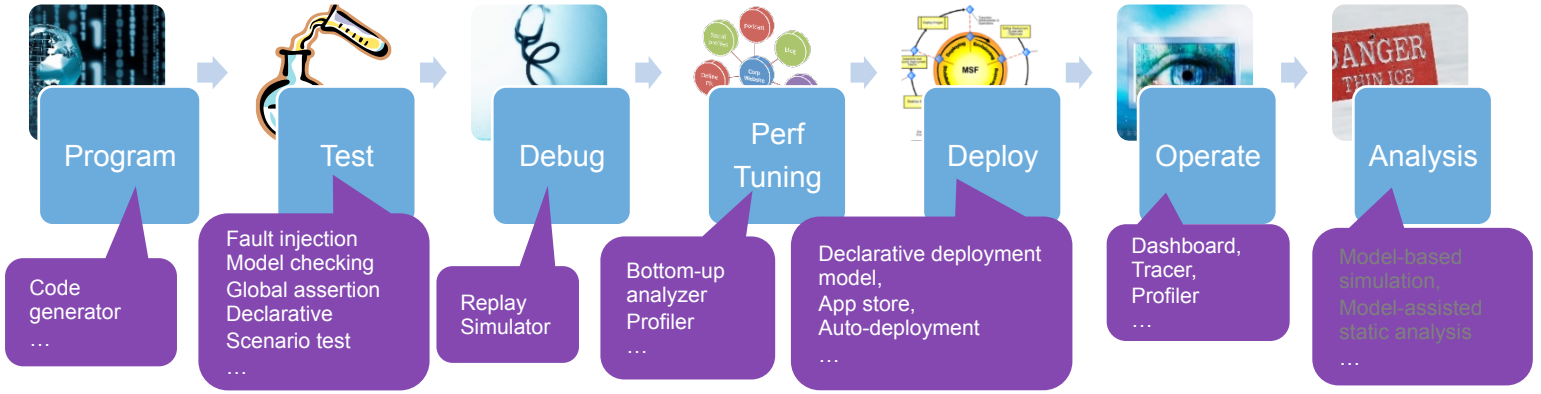
Explicit contract and automated fail-over





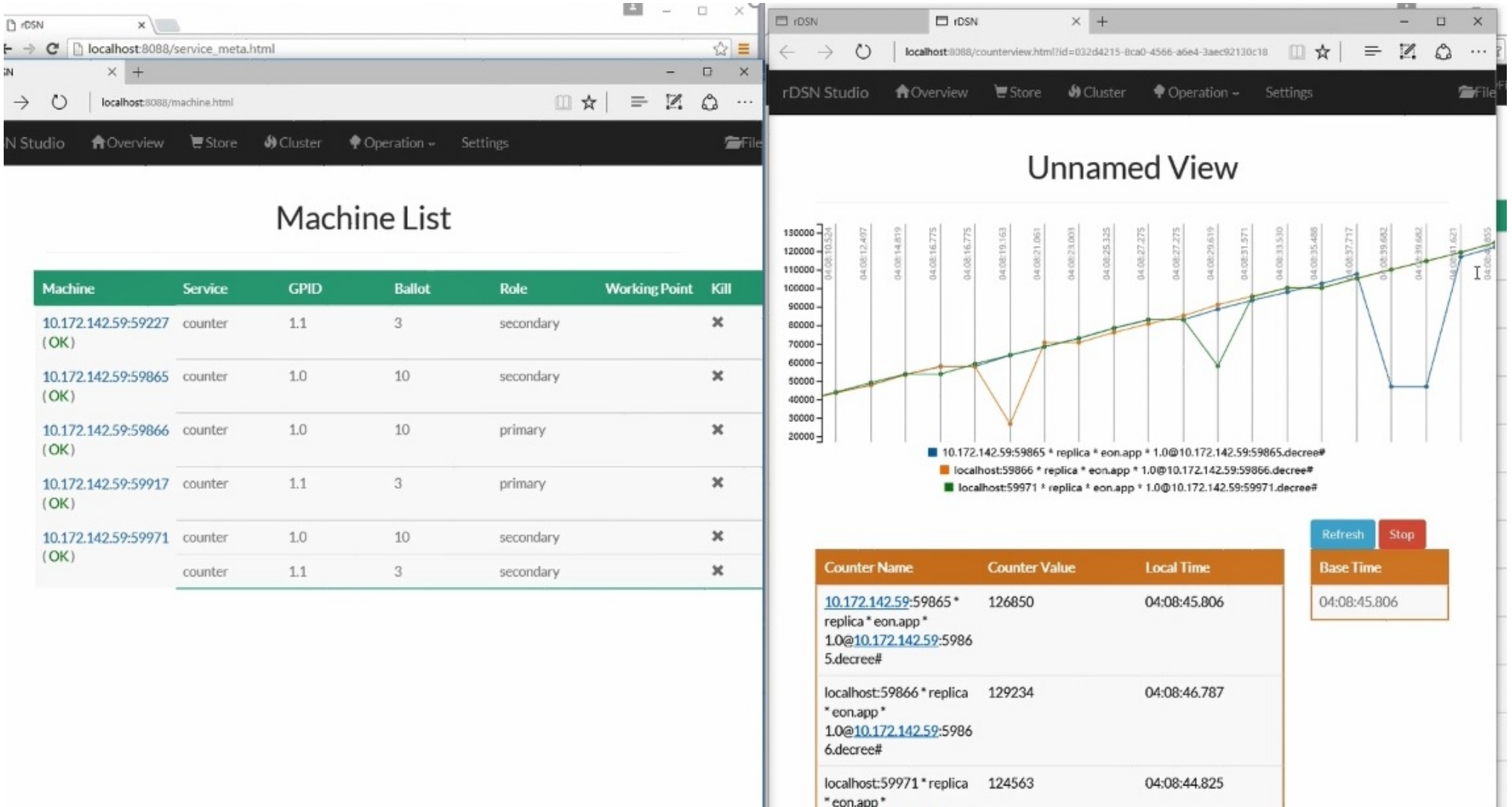


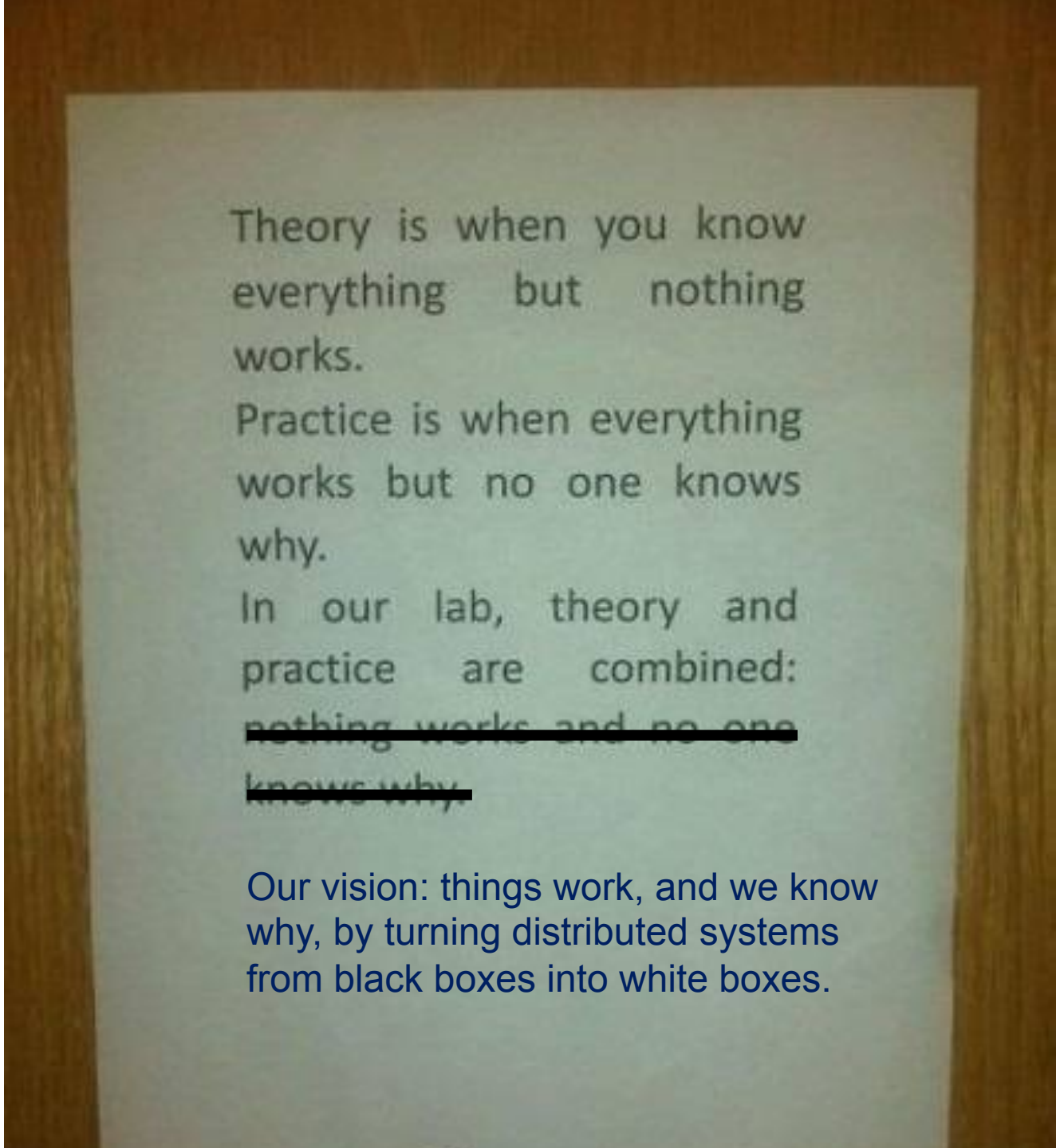
rDSN Studio



Engineering process with integrated tools

applications/frameworks/tools/local libraries development, sharing, discovery, deployment, and operation





Our vision: things work, and we know why, by turning distributed systems from black boxes into white boxes.

Thanks! Questions?

https://github.com/Microsoft/rDSN