PREDICTIVE DATACENTER ANALYTICS WITH STRYMON

Vasia Kalavri
kalavriv@inf.ethz.ch

QCon San Francisco
14 November 2017

Support: AMADEUS FNSNF FONDS NATIONAL SUISSE SCHWEIZERISCHER NATIONALFONDS FONDO NAZIONALE SVIZZERO SWISS NATIONAL SCIENCE FOUNDATION Google

ABOUT ME

@vkalavri

- Postdoc at ETH Zürich
  - Systems Group: https://www.systems.ethz.ch/
- PMC member of Apache Flink
- Research interests
  - Large-scale graph processing
  - Streaming dataflow engines
- Current project
  - Predictive datacenter analytics and management

DATACENTER MANAGEMENT

Solving the Mystery of Link Imbalance: A Metastable Failure State at Scale

Nathan Bronson

This blog post is about code that caused a tricky metastable failure state in Facebook's systems, one that defied explanation for more than two years. It is a great example of interesting things that Facebook's culture of collaboration proved key. Each layer of the system seemed to be working correctly, so it would have been easy for each team to take entrenched positions and blame each other. Instead, we decided that a cross-layer problem would require a cross-layer investigation. We started an internal Facebook group with some network engineers, TAO engineers, and MySQL engineers, and began to look beyond each layer's public abstractions.

https://code.facebook.com/posts/1499322996995183/solving-the-mystery-of-link-imbalance-a-metastable-failure-state-at-scale/

The Register®
Biting the hand that feeds IT

Amadeus booking software outages smack airports across world

“The incident was related to our internal infrastructure. It was triggered by an issue in a faulty switch during network maintenance.

https://www.theregister.co.uk/2017/09/28/amadeus_booking_software_outages_lead_to_global_delayed_flights/

DATACENTER MANAGEMENT

Configuration updates

Network failures

Workload fluctuations

Service deployment

Resource scaling

Software updates

DATACENTER MANAGEMENT

Configuration updates

Network failures

Workload fluctuations

Service deployment

Resource scaling

Software updates

Can we **predict** the effect of changes?

DATACENTER MANAGEMENT

Configuration updates

Network failures

Workload fluctuations

Service deployment

Resource scaling

Software updates

Can we predict the effect of changes?

Can we prevent catastrophic faults?

What–if Analysis:

Predicting outcomes under hypothetical conditions

What–if Analysis:

Predicting outcomes under hypothetical conditions

- How will response time change if we migrate a large service?
- What will happen to link utilization if we change the routing protocol costs?
- Will SLOs will be violated if a certain switch fails?
- Which services will be affected if we change load balancing strategy?

Test deployment?

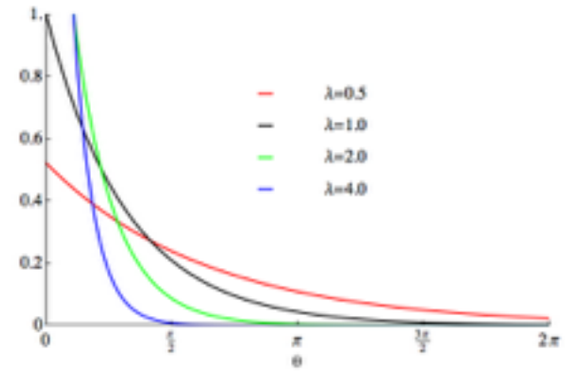
a physical small-scale cluster
to try out configuration changes and what-ifs

- expensive to operate and maintain
- some errors only occur in a large scale!

Analytical model?

infer workload distribution from samples and analytically model system components (e.g. disk failure rate)

Data Center



- hard to develop, large design space to explore
- often inaccurate

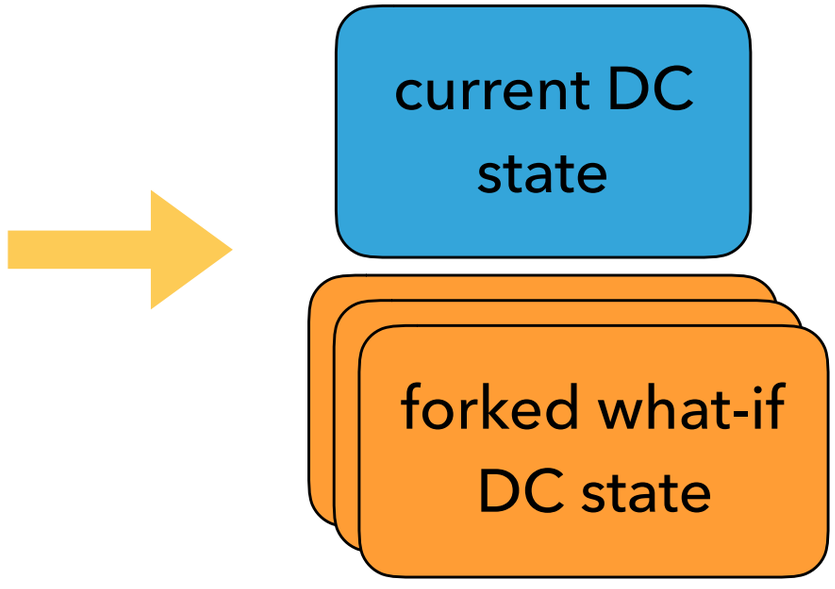
MODERN ENTERPRISE DATACENTERS ARE ALREADY HEAVILY INSTRUMENTED

Trace-driven online simulation

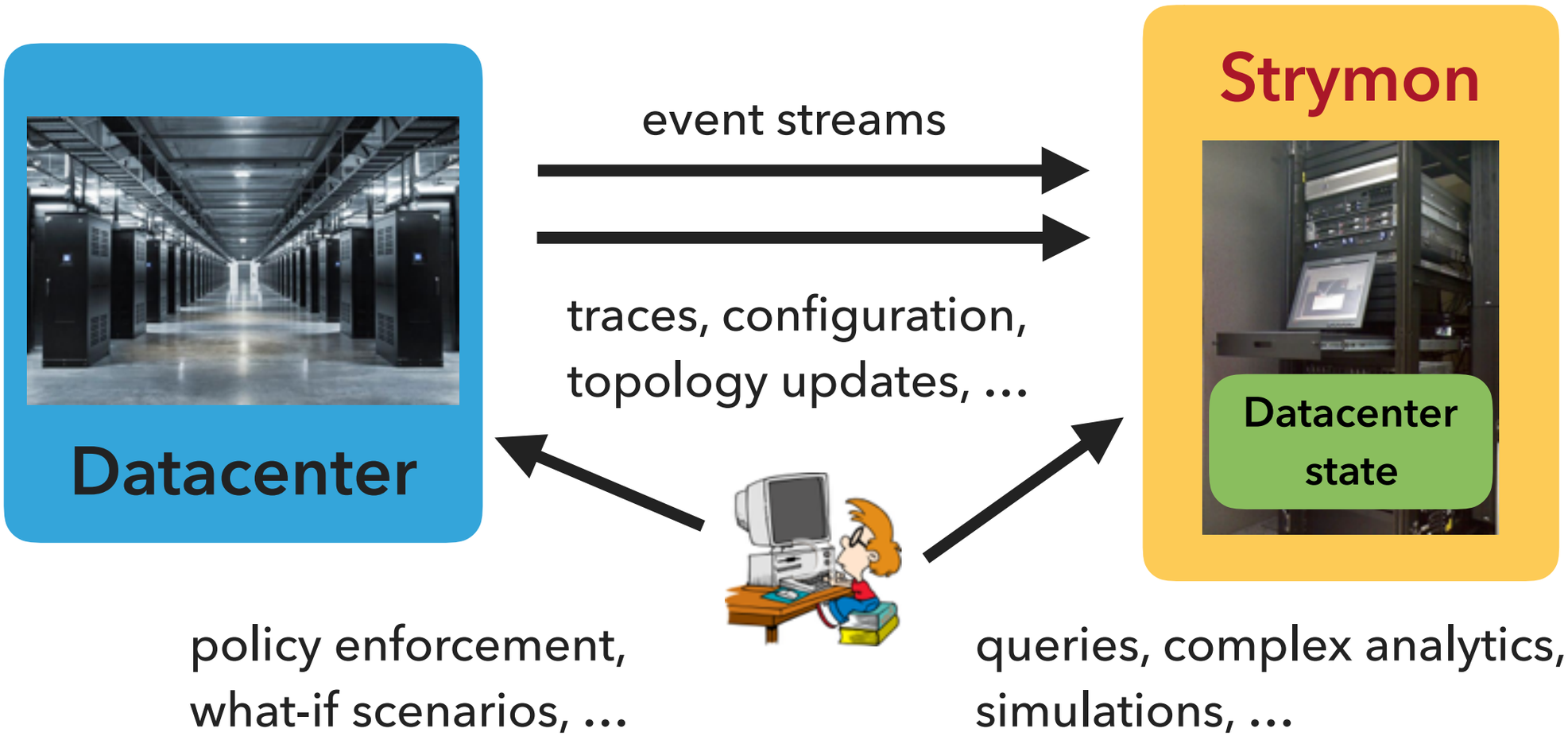
Use existing instrumentation to build a datacenter model

- construct DC state from real events
- simulate the state we cannot directly observe

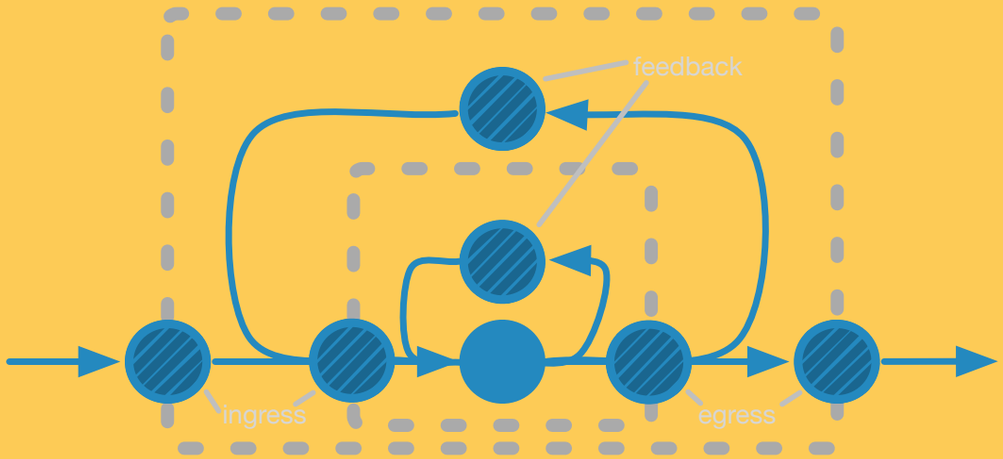
Data Center



STRYMON: ONLINE DATACENTER ANALYTICS AND MANAGEMENT



strymon.systems.ethz.ch



TIMELY DATAFLOW

STRYMON'S OPERATIONAL REQUIREMENTS

- **Low latency**: react quickly to network failures
- **High throughput**: keep up with high stream rates
- **Iterative computation**: complex graph analytics on the network topology
- **Incremental computation**: reuse already computed results when possible
  - e.g. do not recompute forwarding rules after a link update

TIMELY DATAFLOW: STREAM PROCESSING IN RUST

- Data-parallel computations
  - Arbitrary cyclic dataflows
  - Logical timestamps (epochs)
  - Asynchronous execution
  - Low latency

https://github.com/frankmcsherry/timely-dataflow

D. Murray, F. McSherry, M. Isard, R. Isaacs, P. Barham, M. Abadi.
Naiad: A Timely Dataflow System. In SOSP, 2013.

WORDCOUNT IN TIMELY

```
fn main() {
    timely::execute_from_args(std::env::args(), |worker| {

        let mut input = InputHandle::new();
        let mut probe = ProbeHandle::new();
        let index = worker.index();

        worker.dataflow(|scope| {
            input.to_stream(scope)
                .flat_map(|text: String|
                    text.split whitespace()
                        .map(move |word| (word.to_owned(), 1))
                        .collect::<Vec<_>>()
                )
                .aggregate(
                    |_key, val, agg| { *agg += val; },
                    |key, agg: i64| (key, agg),
                    |key| hash_str(key)
                )
                .inspect(|data| println!("seen {:?}", data))
                .probe_with(&mut probe);
        });
         //feed data
         …
    }).unwrap();
}
```

WORDCOUNT IN TIMELY

```
fn main() {
    timely::execute_from_args(std::env::args(), |worker| {

        let mut input = InputHandle::new();
        let mut probe = ProbeHandle::new();
        let index = worker.index();

        worker.dataflow(|scope| {
            input.to_stream(scope)
                .flat_map(|text: String|
                    text.split_whitespace()
                        .map(move |word| (word.to_owned(), 1))
                        .collect::<Vec<_>>()
                )
                .aggregate(
                    |_key, val, agg| { *agg += val; },
                    |key, agg: i64| (key, agg),
                    |key| hash_str(key)
                )
                .inspect(|data| println!("seen {:?}", data))
                .probe_with(&mut probe);
        });
         //feed data
         …
    }).unwrap();
}
```

initialize and run a timely job

WORDCOUNT IN TIMELY

```
fn main() {
    timely::execute_from_args(std::env::args(), |worker| {

        let mut input = InputHandle::new();
        let mut probe = ProbeHandle::new();
        let index = worker.index();

        worker.dataflow(|scope| {
            input.to_stream(scope)
                .flat_map(|text: String|
                    text.split_whitespace()
                        .map(move |word| (word.to_owned(), 1))
                        .collect::<Vec<_>>()
                )
                .aggregate(
                    |_key, val, agg| { *agg += val; },
                    |key, agg: i64| (key, agg),
                    |key| hash_str(key)
                )
                .inspect(|data| println!("seen {:?}", data))
                .probe_with(&mut probe);
        });
        //feed data
        …
    }).unwrap();
}
```

create input and progress handles

WORDCOUNT IN TIMELY

```
fn main() {
    timely::execute_from_args(std::env::args(), |worker| {

        let mut input = InputHandle::new();
        let mut probe = ProbeHandle::new();
        let index = worker.index();

        worker.dataflow(|scope| {
            input.to_stream(scope)
                .flat_map(|text: String|
                    text.split_whitespace()
                        .map(move |wo                    1))
                        .collect::<Vec
                )
                .aggregate(
                    |_key, val, agg| {
                    |key, agg: i64| (key, agg),
                    |key| hash_str(key)
                )
                .inspect(|data| println!("seen {:?}", data))
                .probe_with(&mut probe);
        });
        //feed data
        …
    }).unwrap();
}
```

define the dataflow and its operators

WORDCOUNT IN TIMELY

```
fn main() {
    timely::execute_from_args(std::env::args(), |worker| {

        let mut input = InputHandle::new();
        let mut probe = ProbeHandle::new();
        let index = worker.index();

        worker.dataflow(|scope| {
            input.to_stream(scope)
                .flat_map(|text: String|
                    text.split whitespace()
                        .map(move |word| (word.to_owned(), 1))
                        .collect::<Vec<_>>()
                )
                .aggregate(
                    |_key, val, agg| { *agg += val; },
                    |key, agg: i64| (key, agg),
                    |key| hash_str(key)
                )
                .inspect(|data| println!("s                ata))
                .probe_with(&mut probe);
        });
         //feed data
         …
    }).unwrap();
}
```

watch for progress

WORDCOUNT IN TIMELY

```
fn main() {
    timely::execute_from_args(std::env::args(), |worker| {

        let mut input = InputHandle::new();
        let mut probe = ProbeHandle::new();
        let index = worker.index();

        worker.dataflow(|scope| {
            input.to_stream(scope)
                .flat_map(|text: String|
                    text.split_whitespace()
                        .map(move |word| (word.to_owned(), 1))
                        .collect::<Vec<_>>()
                )
                .aggregate(
                    |_key, val, agg| { *agg += val; },
                    |key, agg: i64| (key, agg),
                    |key| hash_str(key)
                )
                .inspect(|data| println!("seen {:?}", data))
                .probe_with(&mut probe);
        });
         //feed data
         …
    }).unwrap();
}
```

a few Rust peculiarities to get used to :-)

PROGRESS TRACKING

A distributed protocol that allows operators reason about the possibility of receiving data

- All tuples bear a logical timestamp (*think event time*)
- To send a timestamped tuple, an operator **must** hold a capability for it
- Workers broadcast progress changes to other workers
- Each worker independently determines progress

MAKING PROGRESS

```
fn main() {
    timely::execute_from_args(std::env::args(), |worker| {
        …
        …
        …
            for round in 0..10 {
                input.send(("round".to_owned(), 1));
                input.advance_to(round + 1);
                while probe.less_than(input.time()) {
                    worker.step();
                }
            }
    }).unwrap();
}
```

MAKING PROGRESS

```
fn main() {
    timely::execute_from_args(std::env::args(), |worker| {
        …
        …
        …
            for round in 0..10 {
                input.send(("round".to_owned(), 1));
                input.advance_to(round + 1);
                while probe.less_than(input.time()) {
                    worker.step();
                }
            }
    }).unwrap();
}
```

push data to the input stream

MAKING PROGRESS

```
fn main() {
    timely::execute_from_args(std::env::args(), |worker| {
        …
        …
        …
            for round in 0..10 {
                input.send(("round".to_owned()
                input.advance_to(round + 1);
                while probe.less_than(input.ti
                    worker.step();
                }
            }
    }).unwrap();
}
```

advance the input epoch

MAKING PROGRESS

```
fn main() {
    timely::execute_from_args(std::env::args(), |worker| {
        …
        …
        …
            for round in 0..10 {
                input.send(("round".to_owned(), 1));
                input.advance_to(round + 1);
                while probe.less_than(input.time()) {
                    worker.step();
                }
            }
    }).unwrap();
}
```

do work while there's still data for this epoch

TIMELY ITERATIONS

```
timely::example(|scope| {

    let (handle, stream) = scope.loop_variable(100, 1);
    (0..10).to_stream(scope)
           .concat(&stream)
           .inspect(|x| println!("seen: {:?}", x))
           .connect_loop(handle);
});
```

TIMELY ITERATIONS

```
timely::example(|scope| {

    let (handle, stream) = scope.loop_variable(100, 1);
    (0..10).to_stream(scope)
           .concat(&stream)
           .inspect(|x| println!("seen: {:?}",
           .connect_loop(handle);
});
```

loop 100 times at most

advance timestamps by 1 in each iteration

create the feedback loop

TIMELY ITERATIONS

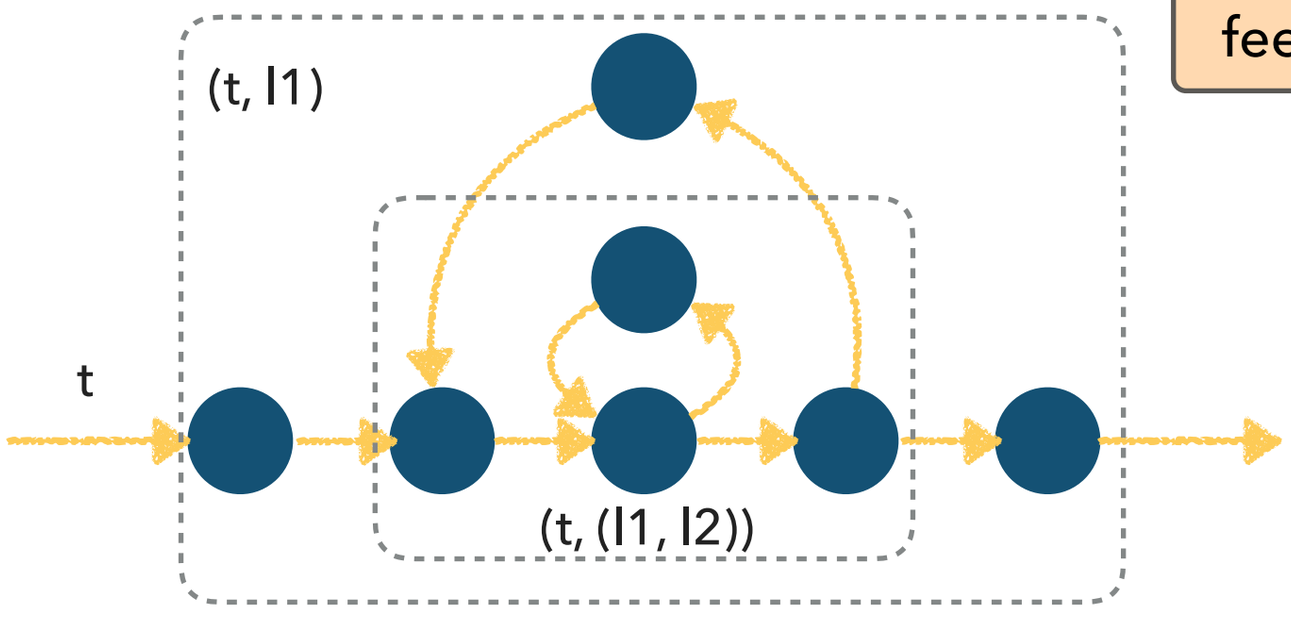
```
timely::example(|scope| {

    let (handle, stream) = scope.loop_variable(100, 1);
    (0..10).to_stream(scope)
           .concat(&stream)
           .inspect(|x| println!("seen: {:?}",
           .connect_loop(handle);
});
```

loop 100 times at most

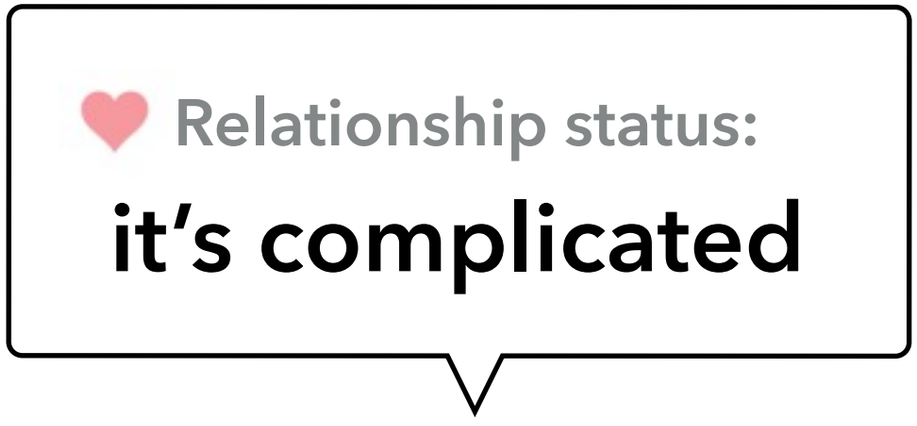
advance timestamps by 1 in each iteration

create the feedback loop



TIMELY & I

TIMELY & I



TIMELY & I

Relationship status:

it's complicated

performance

debugging

incremental computation

fault-tolerance

expressiveness

APIs & libraries

deployment

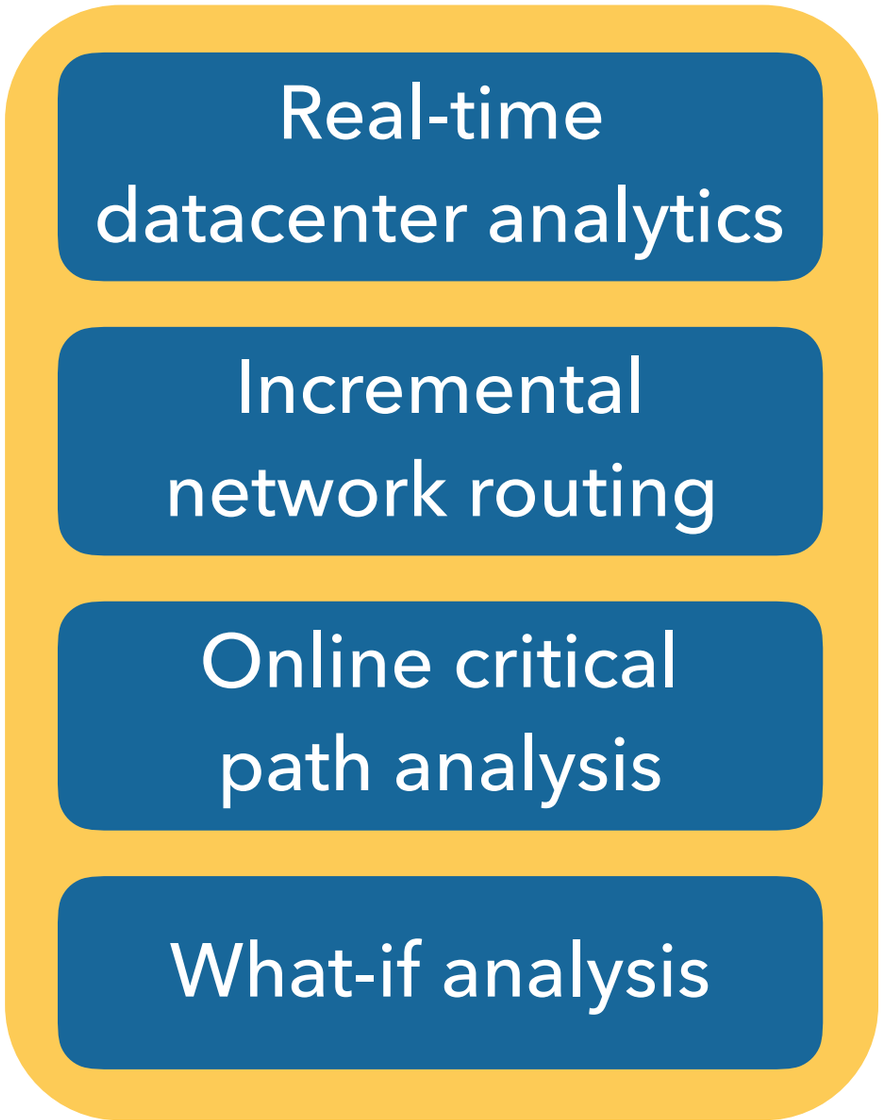
ecosystem

performance

STRYMON

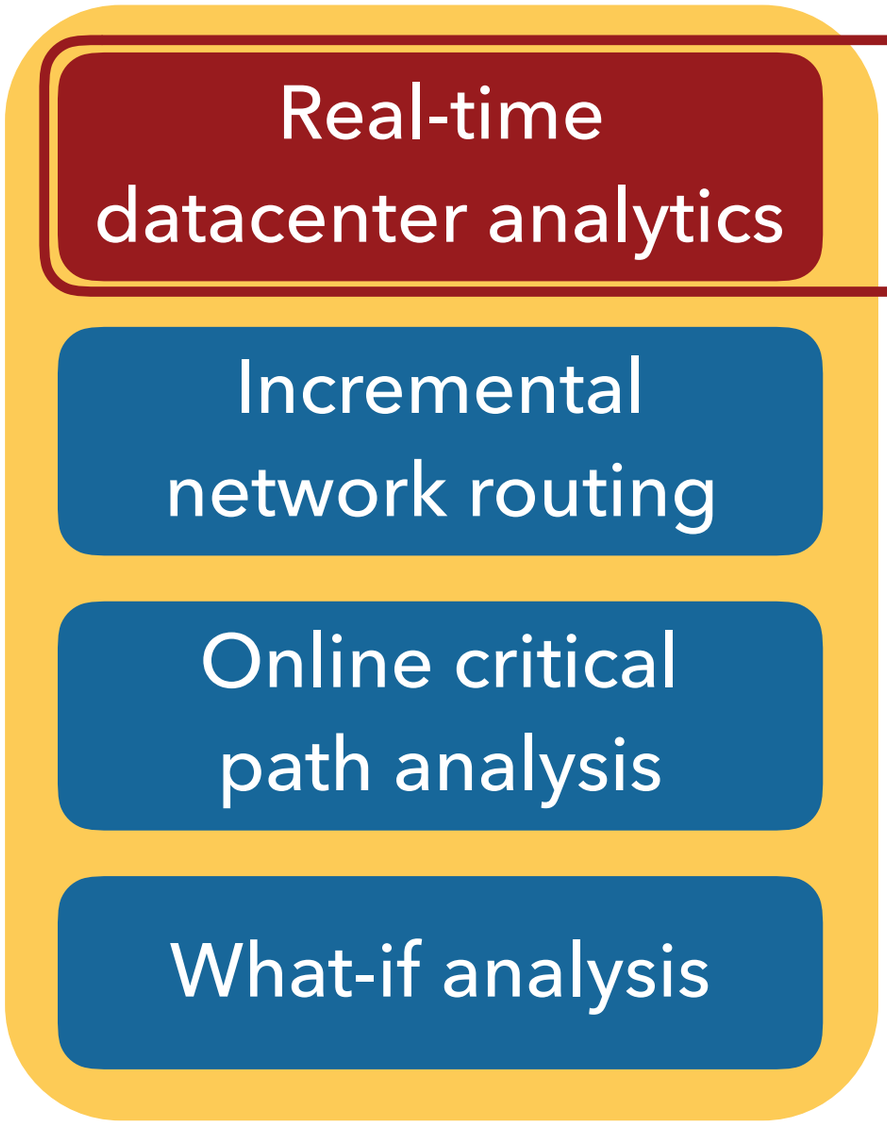
USE-CASES

Strymon



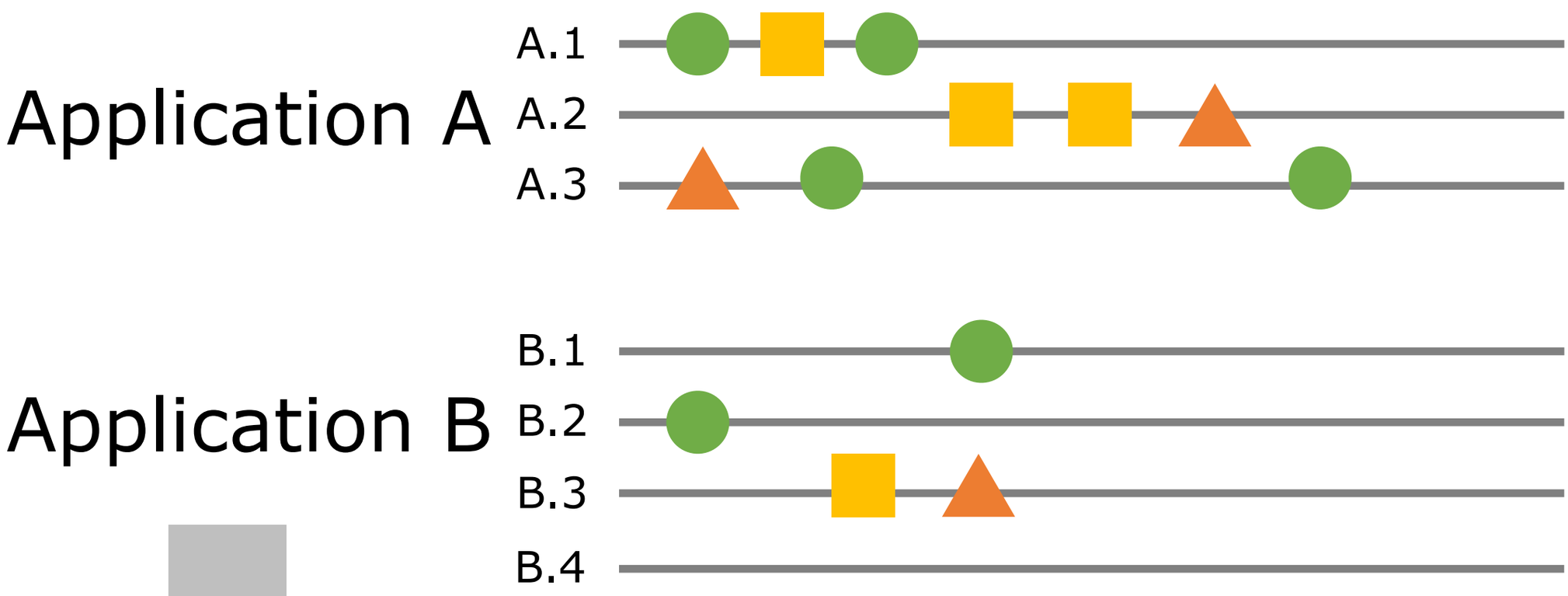
- Query and analyze state online
- Control and enforce configuration
- Understand performance
- Simulate what-if scenarios

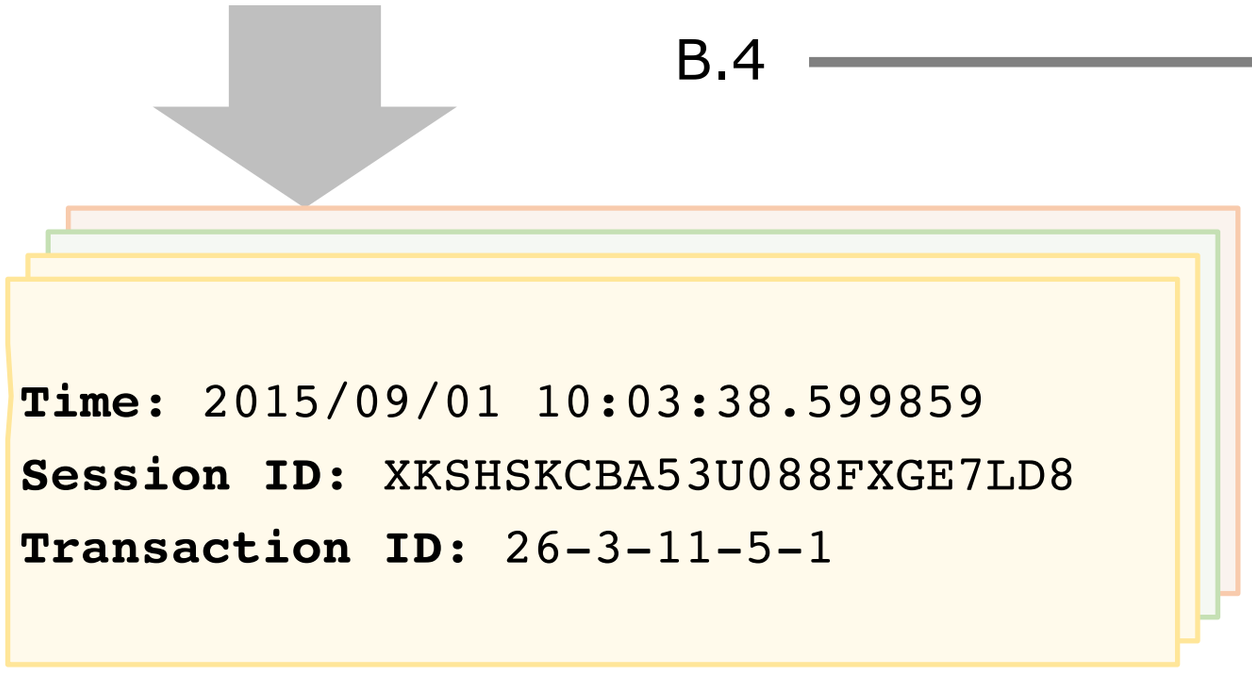
Strymon

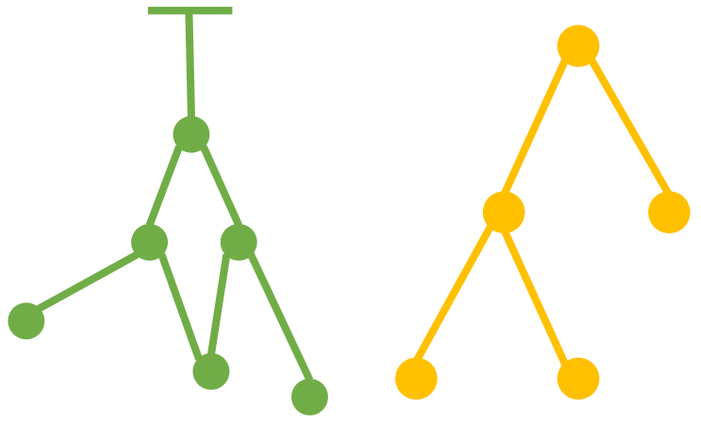


- Query and analyze state online
- Control and enforce configuration
- Understand performance
- Simulate what-if scenarios

RECONSTRUCTING USER SESSIONS

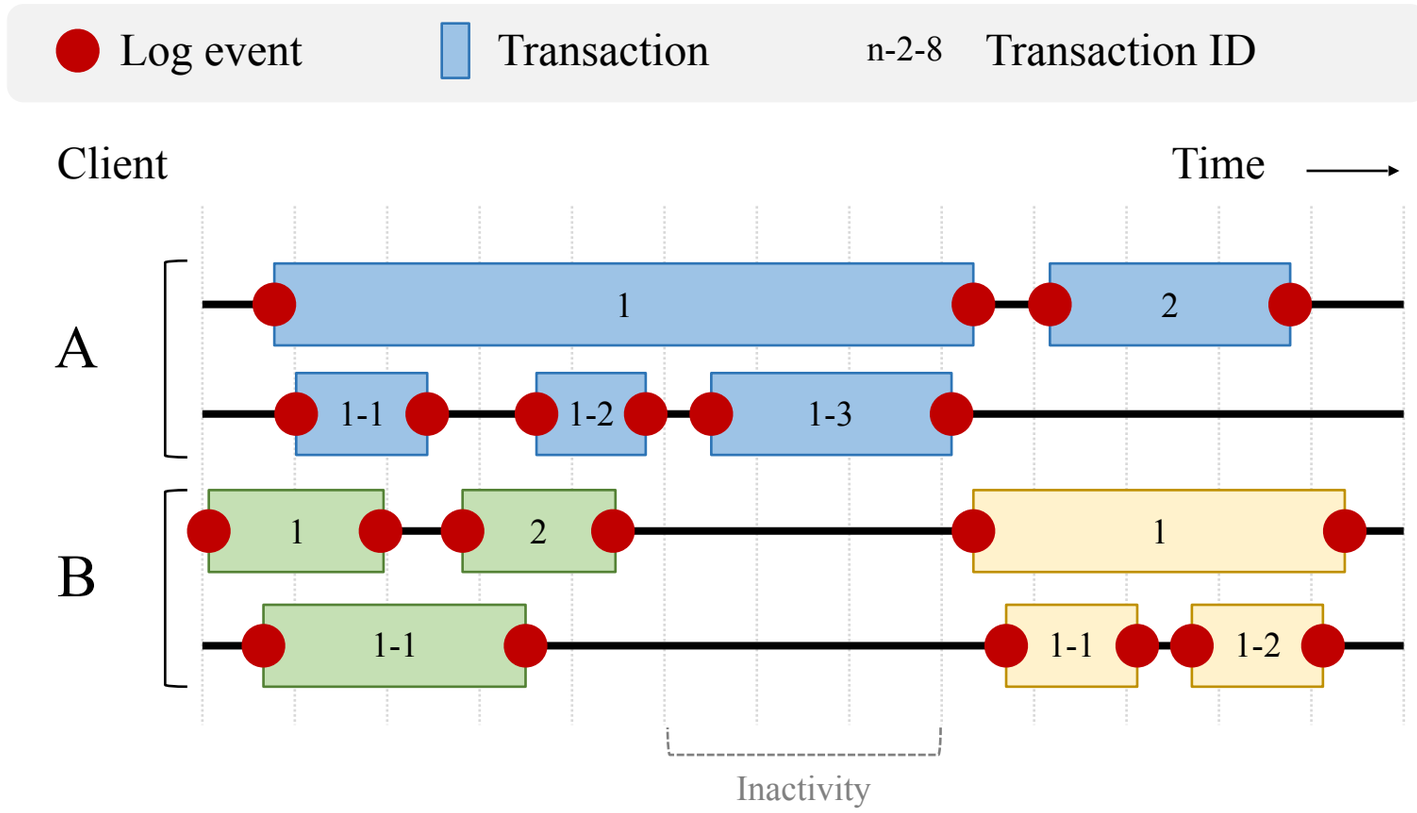






PERFORMANCE RESULTS

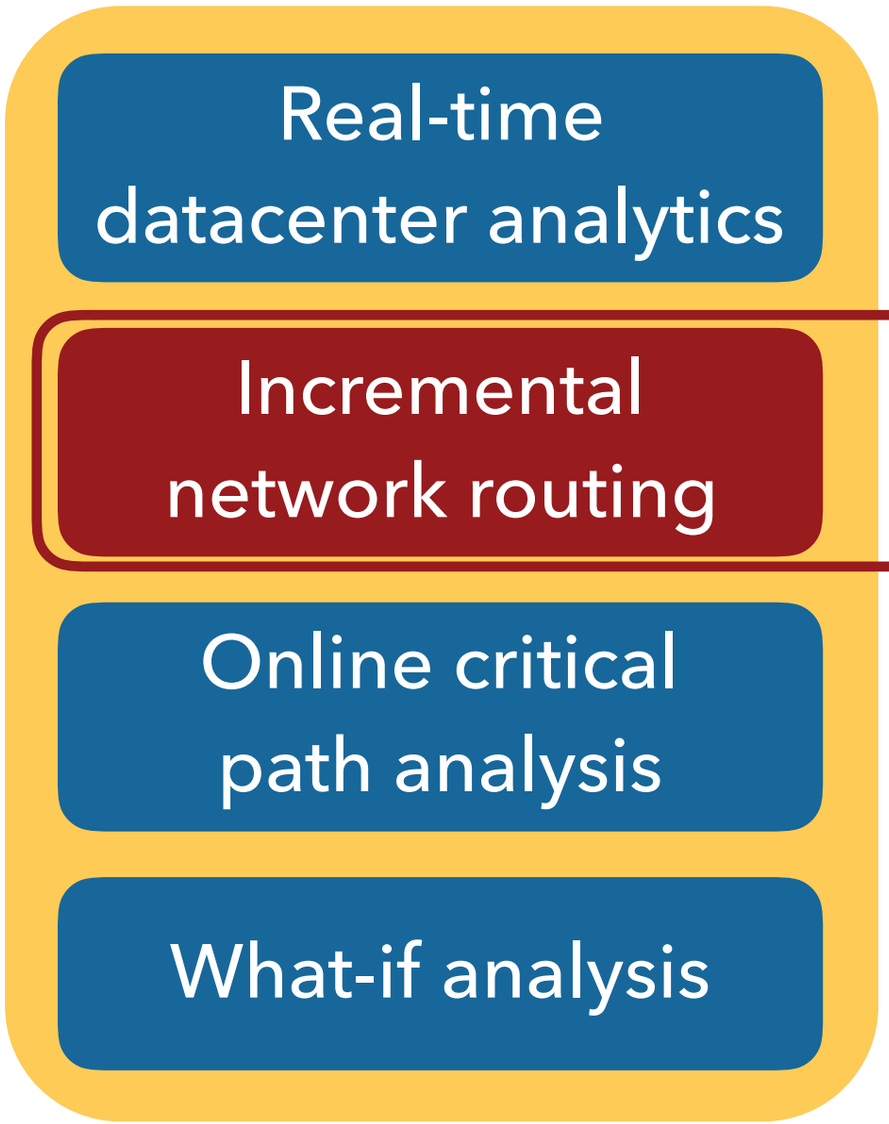
- Logs from **1263 streams** and **42 servers**
- **1.3 million events/s** at **424.3 MB/s**
- **26ms** per epoch vs. **2.1s** per epoch with Flink



More Results:

Zaheer Chothia, John Liagouris, Desislava Dimitrova, and Timothy Roscoe.
Online Reconstruction of Structural Information from Datacenter Logs. (EuroSys '17).

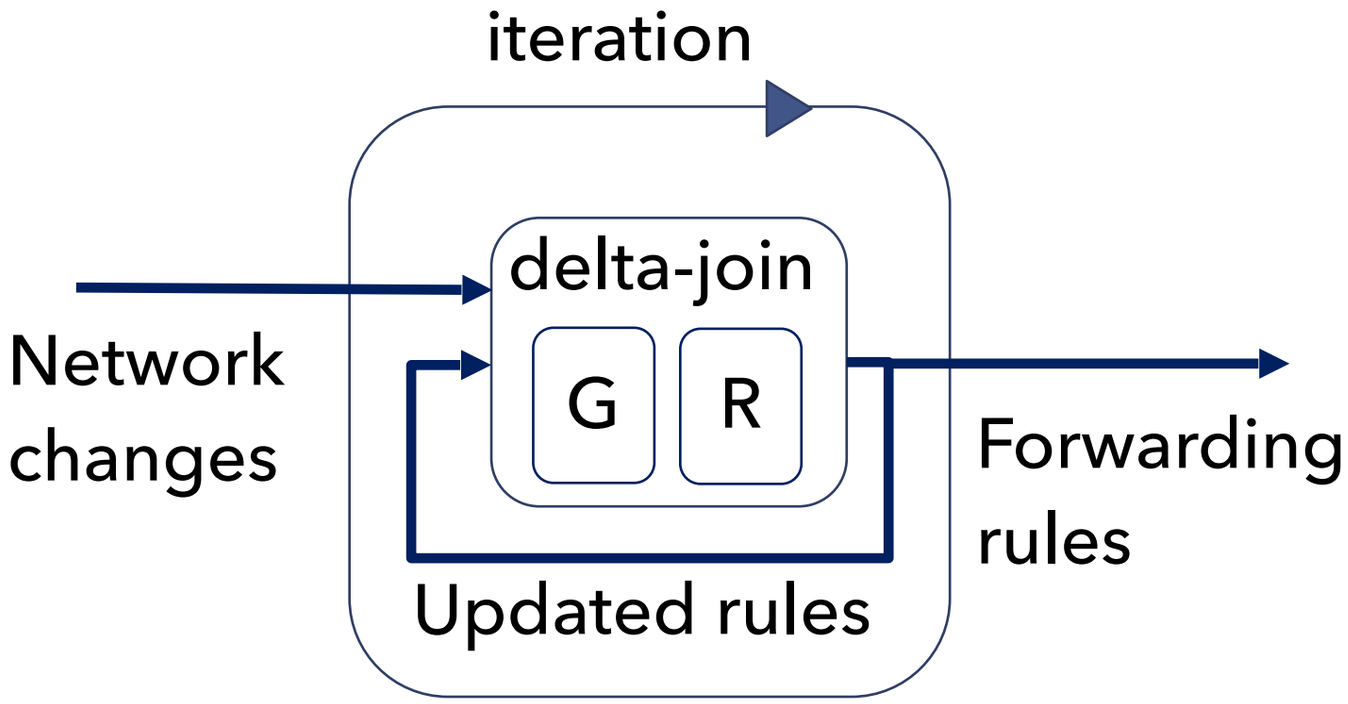
Strymon



- Query and analyze state online
- Control and enforce configuration
- Understand performance
- Simulate what-if scenarios

ROUTING AS A STREAMING COMPUTATION

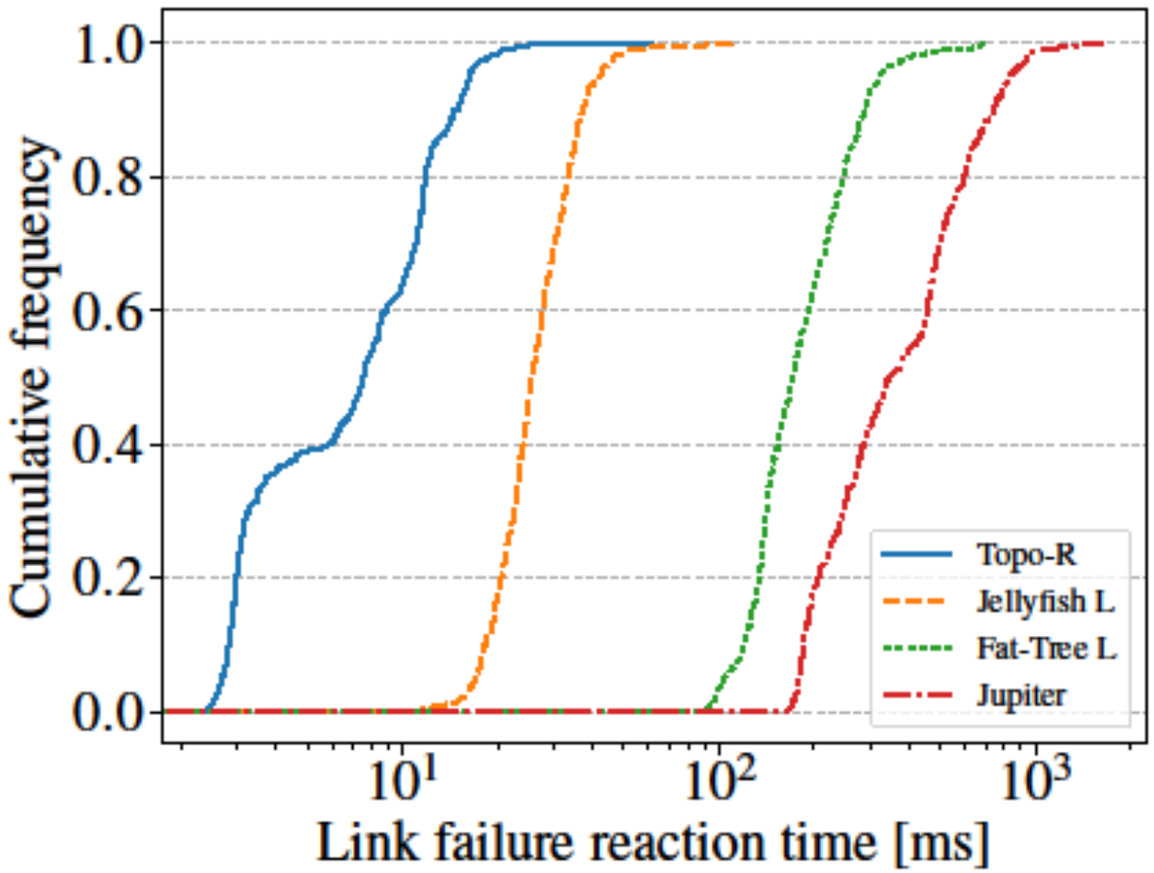
- Compute APSP and keep forwarding rules as operator state
- Flow requests translate to lookups on that state
- Network updates cause re-computation of affected rules only



REACTION TO LINK FAILURES

Topology	Hosts	Switches	Ports	Links
Jellyfish	27648	1280	48	6912
Fat-tree	27648	2880	48	55396
Topo-R	19404	546	*	917
Jupiter	98304	5632	64	87040

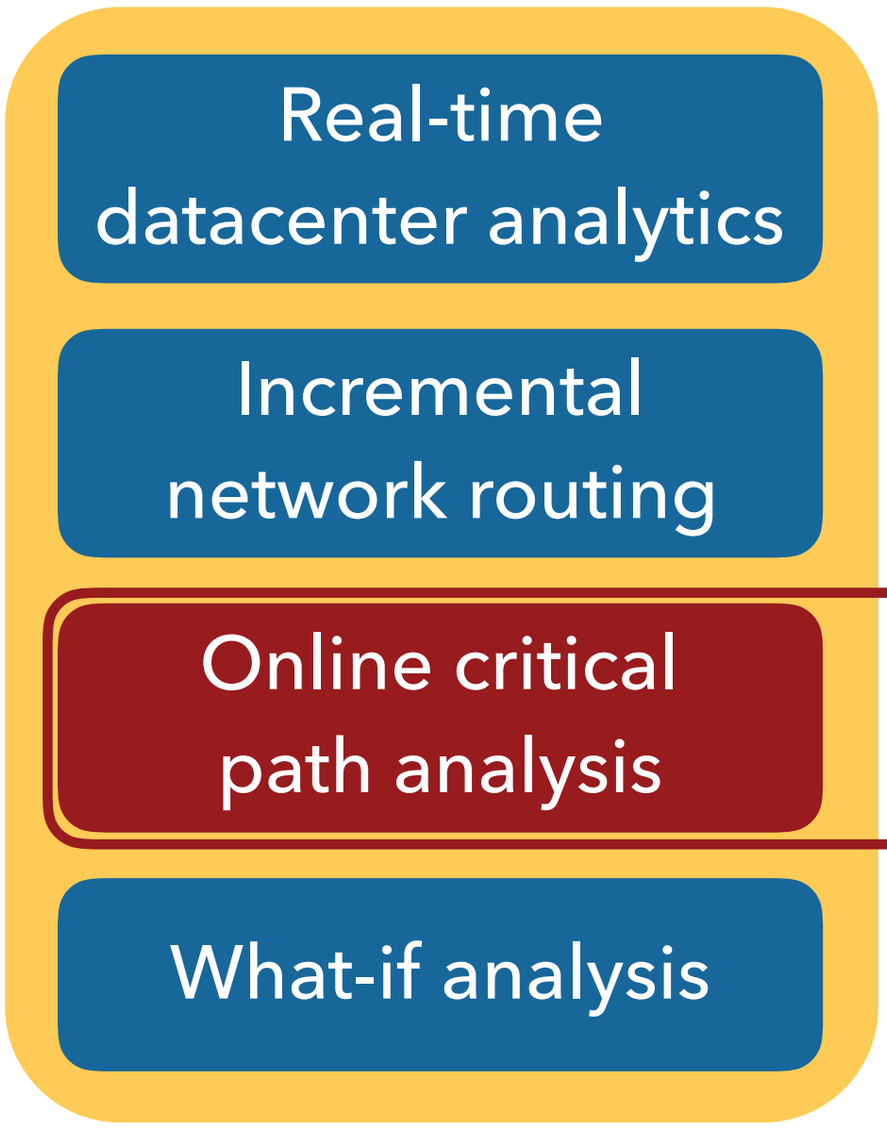
- Fail random link
- 500 individual runs
- 32 threads



More Results:

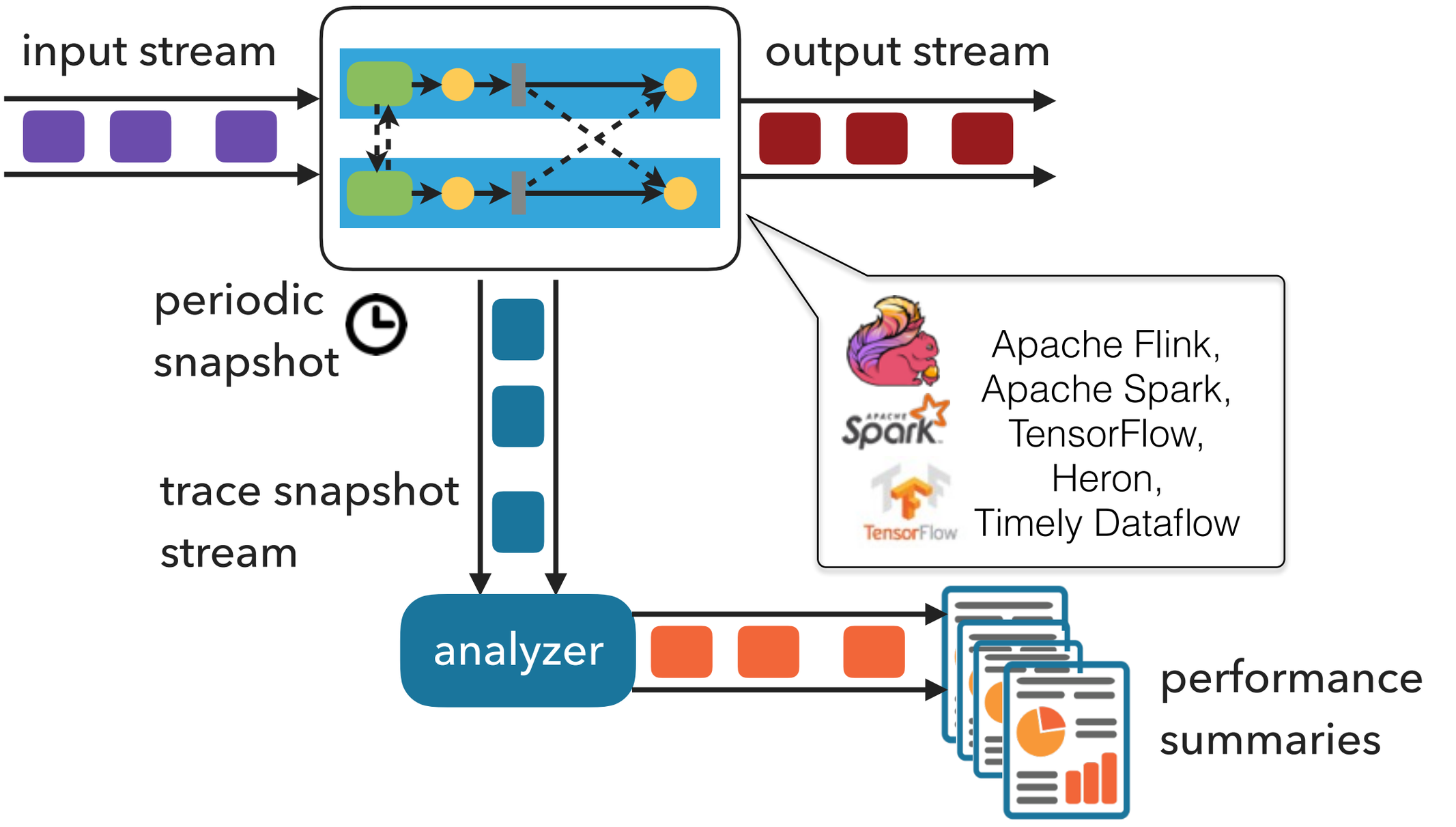
Desislava C. Dimitrova et. al.
Quick Incremental Routing Logic for Dynamic Network Graphs. SIGCOMM Posters and Demos '17

Strymon

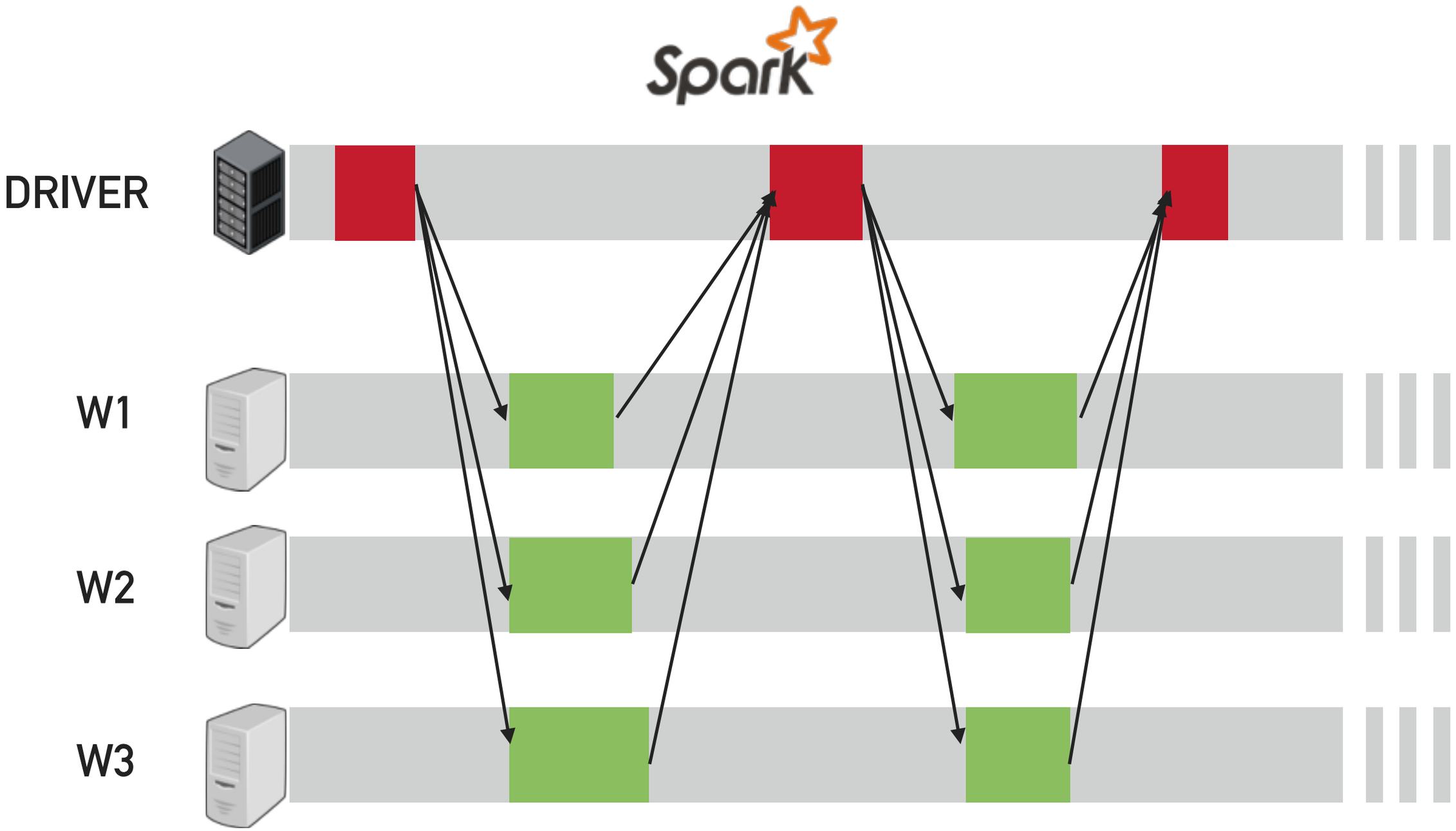


- Query and analyze state online
- Control and enforce configuration
- Understand performance
- Simulate what-if scenarios

ONLINE CRITICAL PATH ANALYSIS

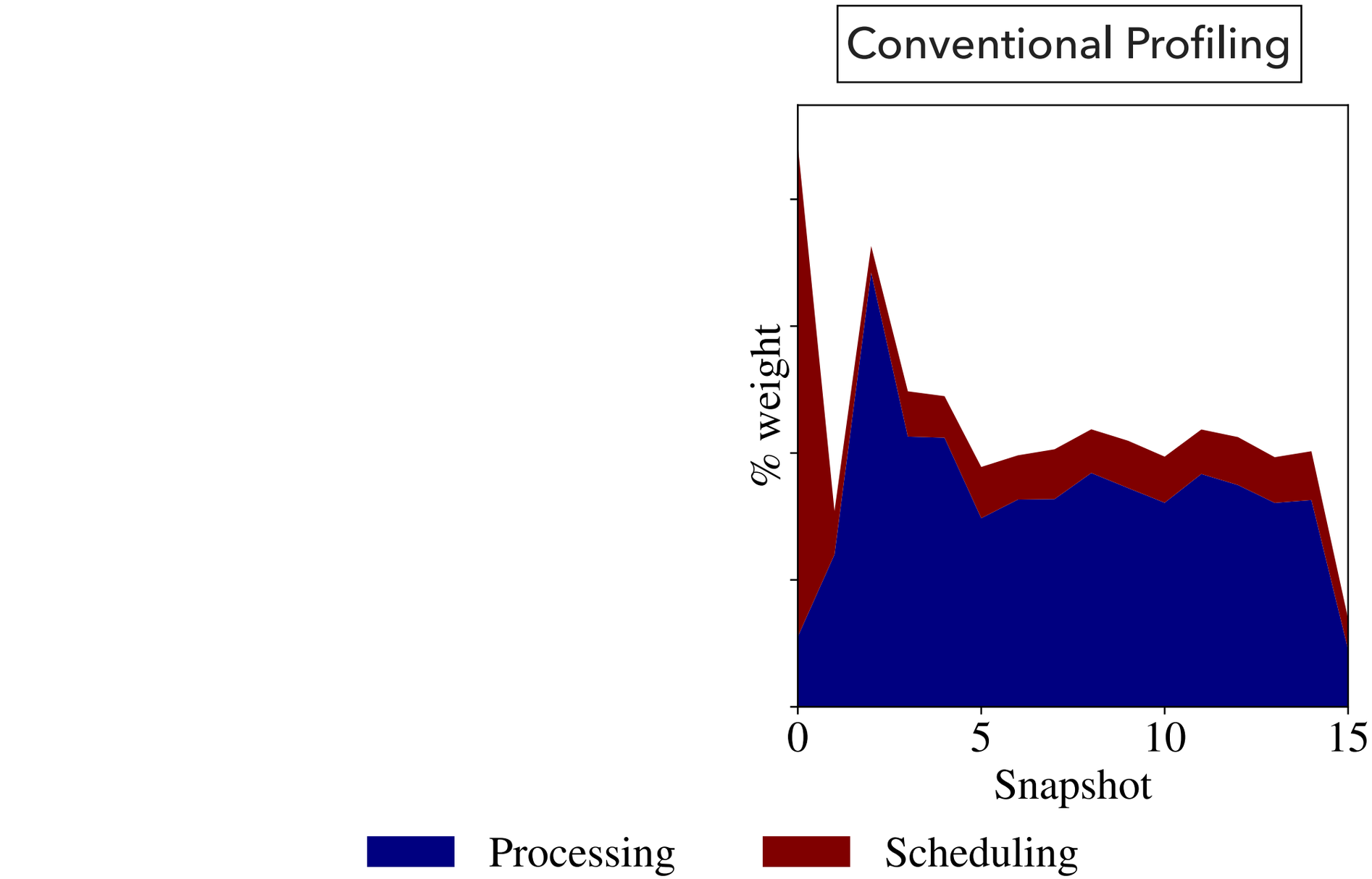


TASK SCHEDULING IN APACHE SPARK



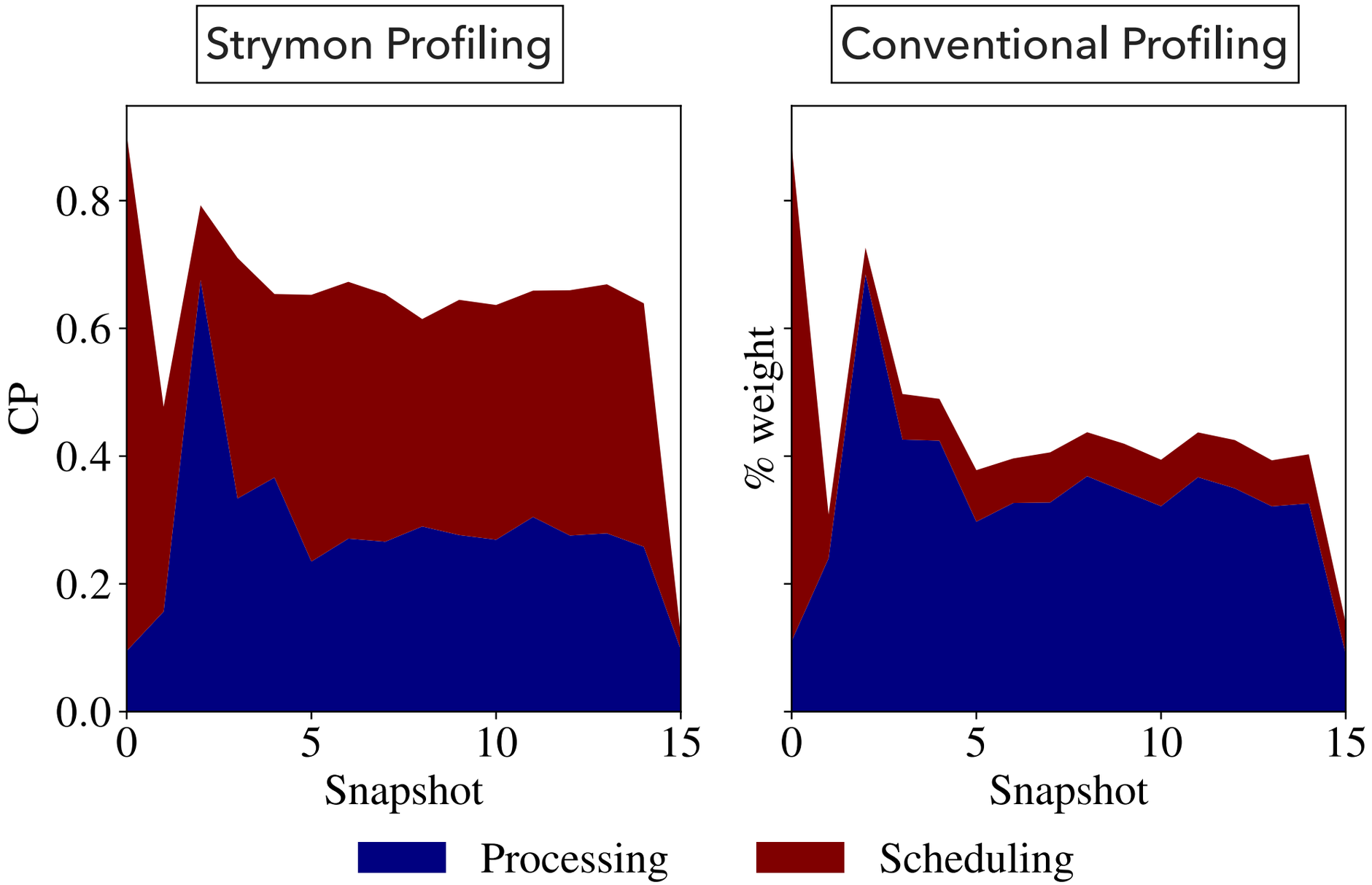
Venkataraman, Shivaram, et al. "Drizzle: Fast and adaptable stream processing at scale." Spark Summit (2016).

SCHEDULING BOTTLENECK IN APACHE SPARK



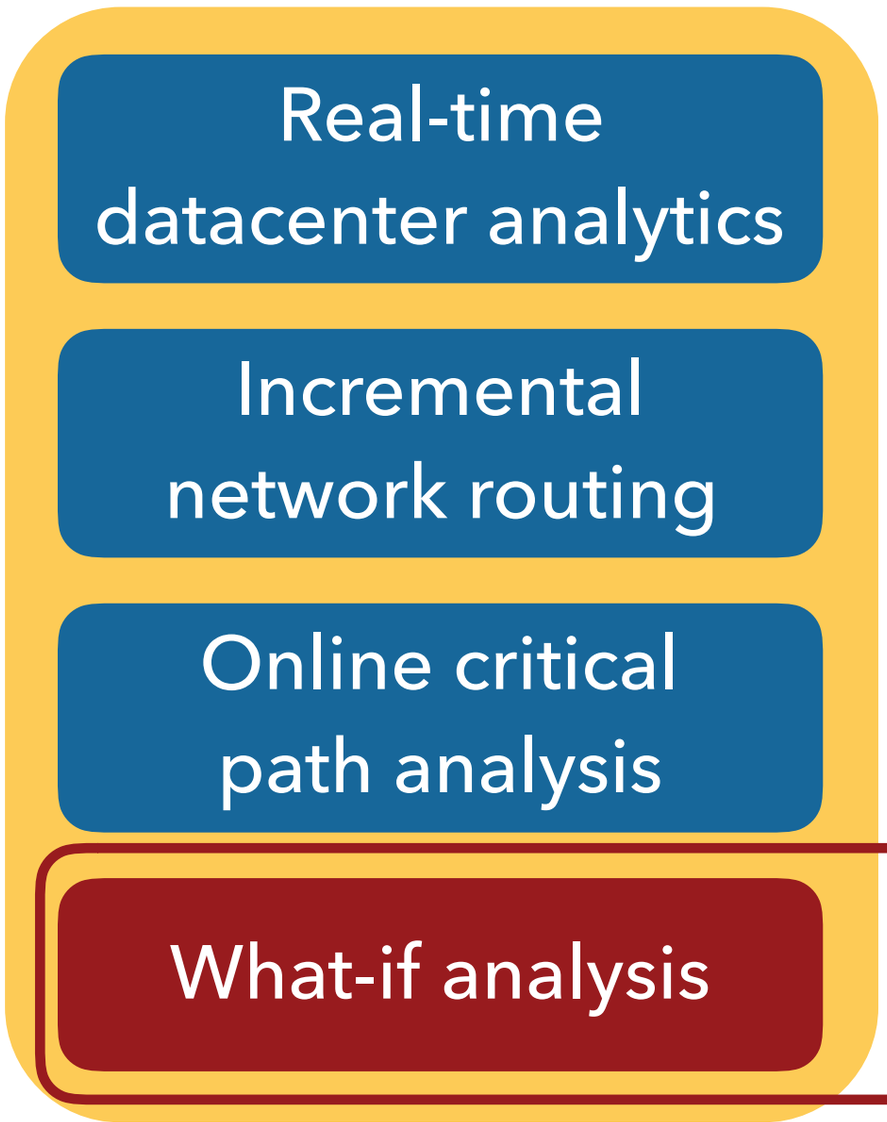
Apache Spark: Yahoo! Streaming Benchmark, 16 workers, 8s snapshots

SCHEDULING BOTTLENECK IN APACHE SPARK



Apache Spark: Yahoo! Streaming Benchmark, 16 workers, 8s snapshots

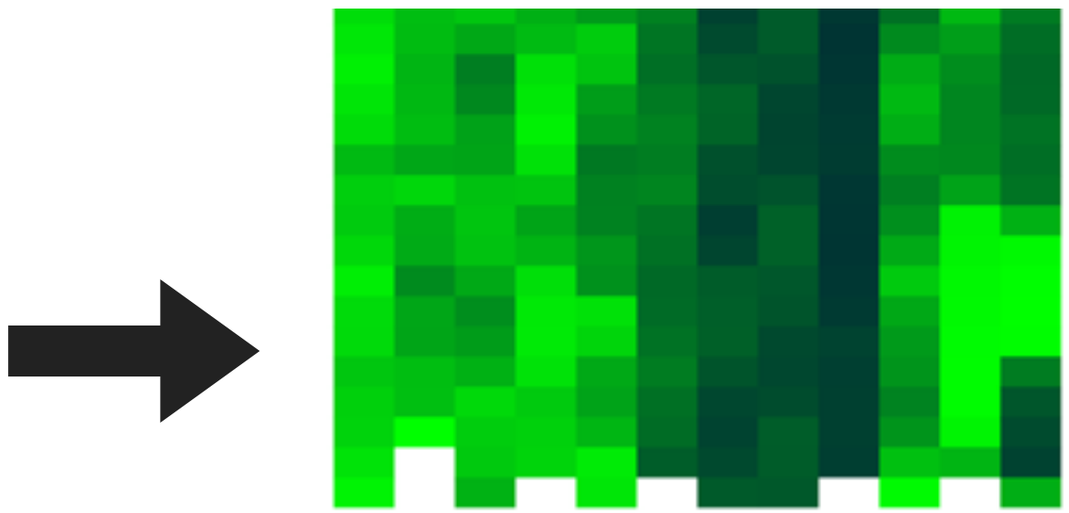
Strymon



- Query and analyze state online
- Control and enforce configuration
- Understand performance
- Simulate what-if scenarios

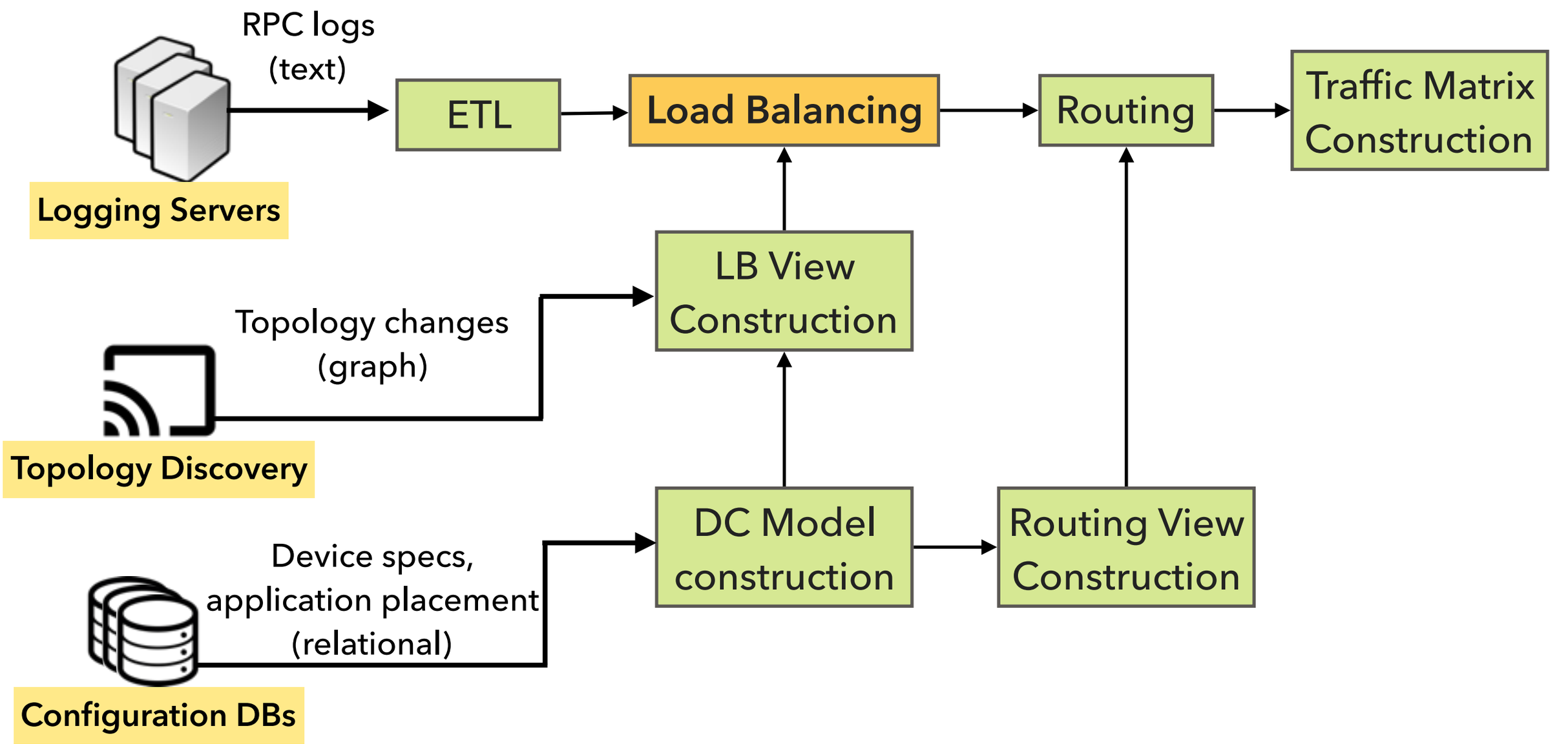
EVALUATING LOAD BALANCING STRATEGIES

- 13k services
- ~100K user requests/s
- OSPF routing
- Weighted Round-Robin load balancing



Traffic matrix simulation under topology and load balancing changes

WHAT-IF DATAFLOW



STRYMON DEMO

Strymon 0.1.0 has been released!

Try it out:

https://strymon-system.github.io

https://github.com/strymon-system

Send us feedback:

strymon-users@lists.inf.ethz.ch

THE STRYMON TEAM & FRIENDS

Prof. Timothy Roscoe

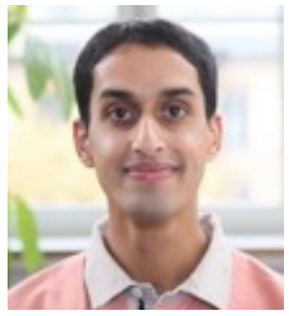
Desislava Dimitrova

Moritz Hoffmann

Andrea Lattuada

Frank McSherry

John Liagouris



Zaheer Chothia

Sebastian Wicki

Vasiliki Kalavri

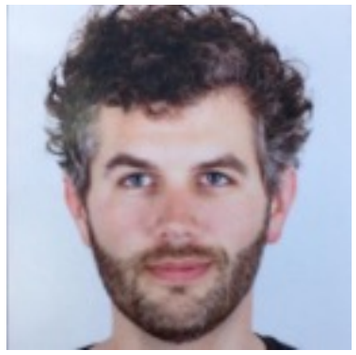
THE STRYMON TEAM & FRIENDS

Prof. Timothy Roscoe

Desislava Dimitrova

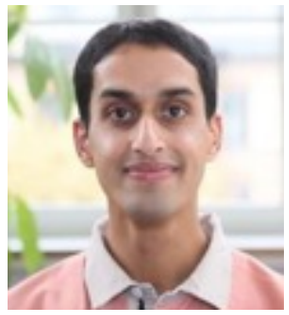
Moritz Hoffmann

Andrea Lattuada



Frank McSherry

John Liagouris



Zaheer Chothia

Sebastian Wicki

Vasiliki Kalavri

?

?

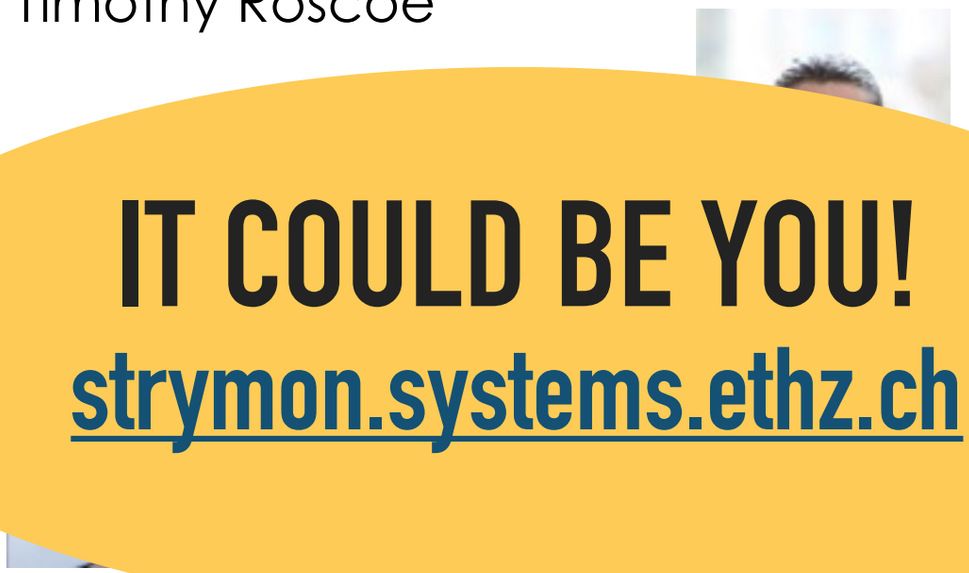
THE STRYMON TEAM & FRIENDS

Prof. Timothy Roscoe

Desislava Dimitrova

Moritz Hoffmann

Andrea Lattuada

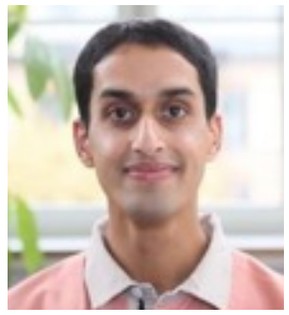


Frank McSherry

IT COULD BE YOU!

strymon.systems.ethz.ch

Vasiliki Kalavri



Zaheer Chothia

Sebastian Wicki

?

?

PREDICTIVE DATACENTER ANALYTICS WITH STRYMON

Vasia Kalavri
kalavriv@inf.ethz.ch

QCon San Francisco
14 November 2017

Support: AMADEUS FNSNF FONDS NATIONAL SUISSE SCHWEIZERISCHER NATIONALFONDS FONDO NAZIONALE SVIZZERO SWISS NATIONAL SCIENCE FOUNDATION Google