Using Distributed Trace for End-to-End Latency Metrics

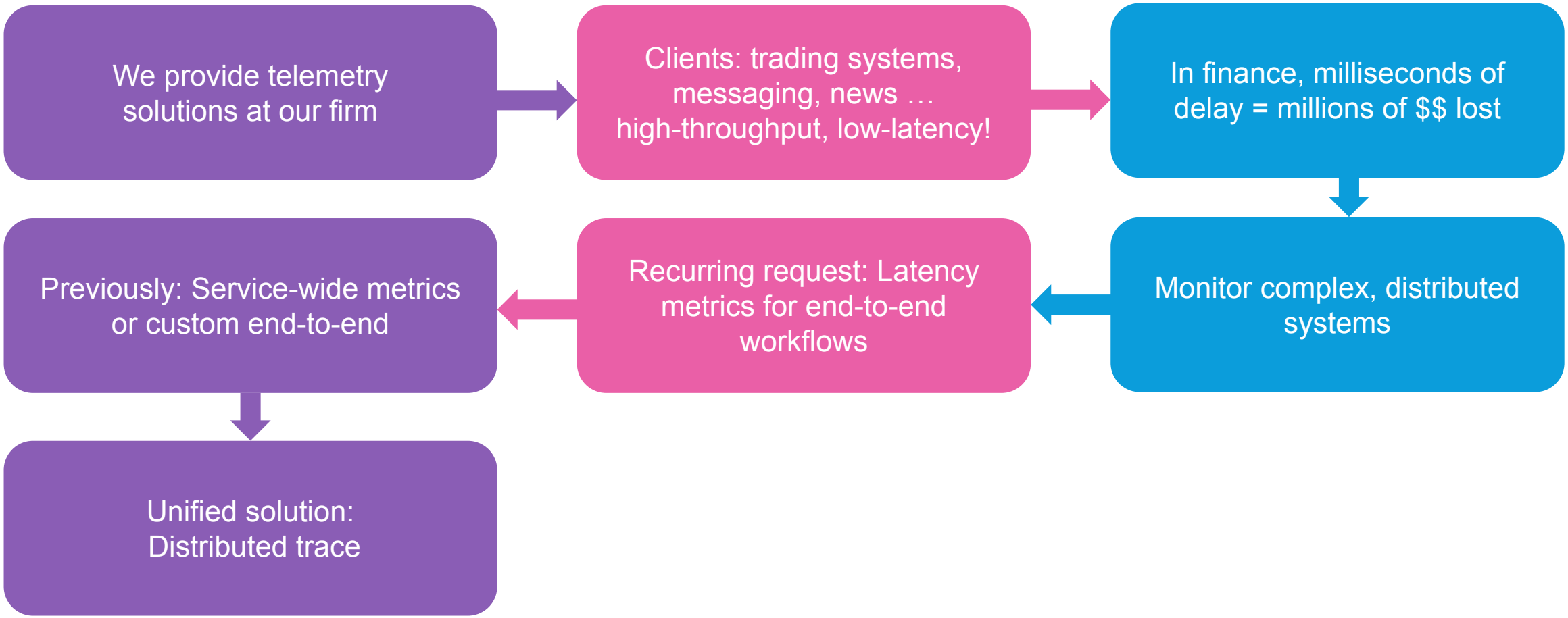
KUSHA MAHARSHI

Cppcon
The C++ Conference

2025

September 13 - 19

Storytime



TechAtBloomberg.com

Bloomberg

Engineering

Storytime

We provide telemetry solutions at our firm

Clients: trading systems, messaging, news … high-throughput, low-latency!

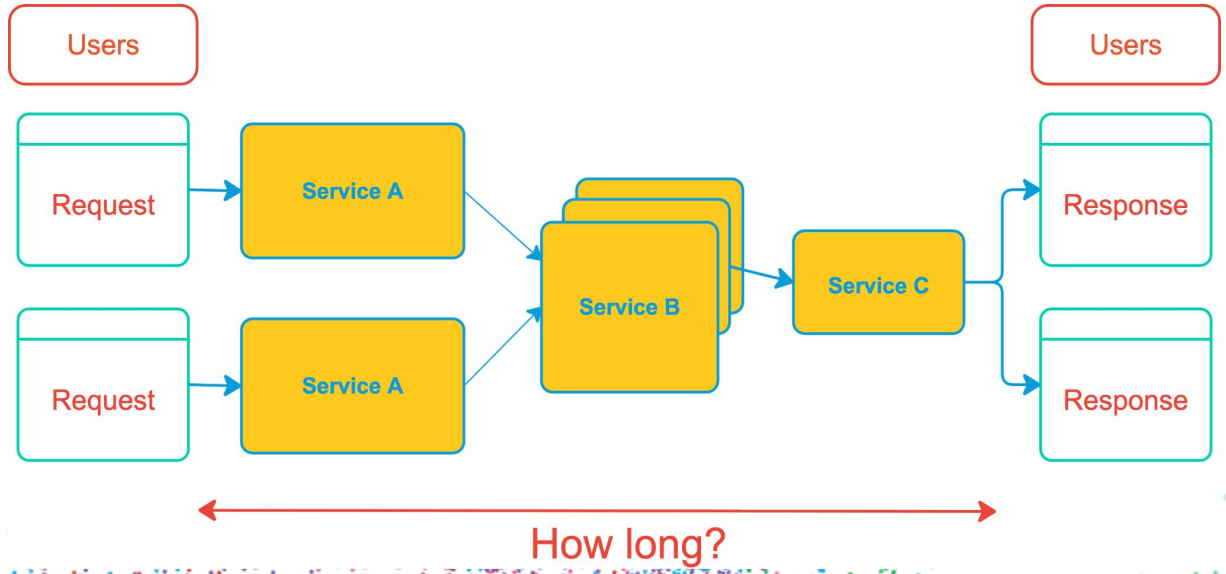
In finance, milliseconds of delay = millions of $$ lost

Monitor complex, distributed systems

Recurring request: Latency metrics for end-to-end workflows

Previously: Service-wide metrics or custom end-to-end

Unified solution: Distributed trace

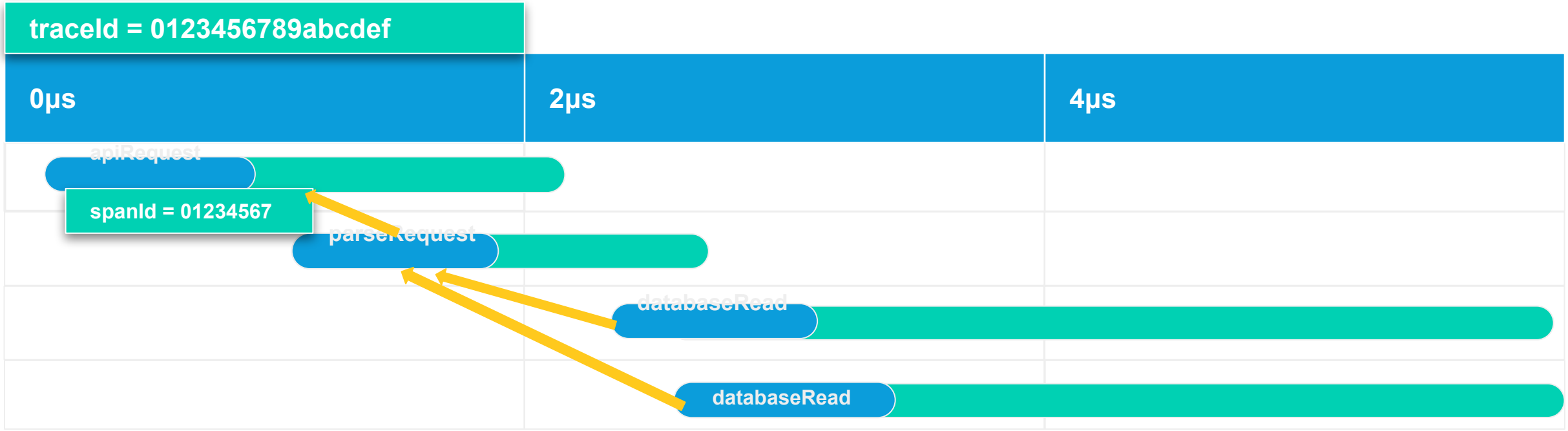


TechAtBloomberg.com

Bloomberg

Engineering

What Is Distributed Tracing?

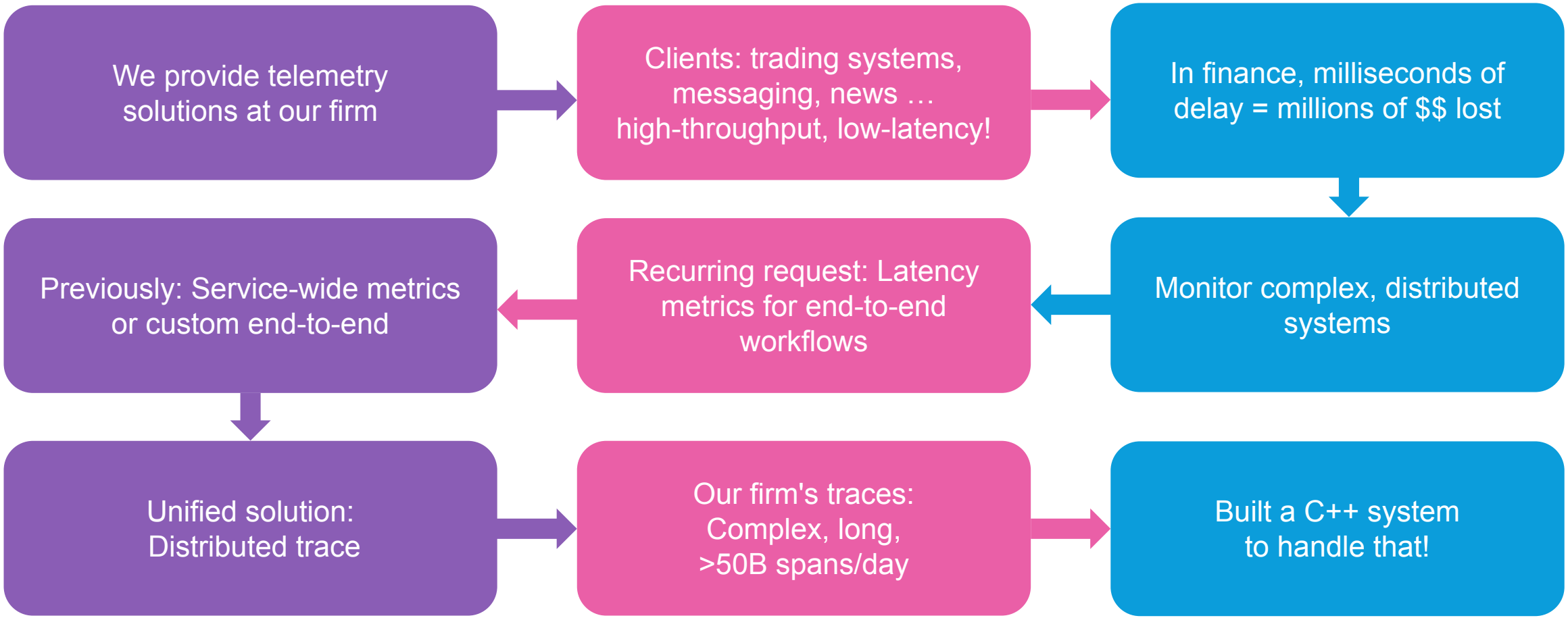


TechAtBloomberg.com

Bloomberg

Engineering

Storytime

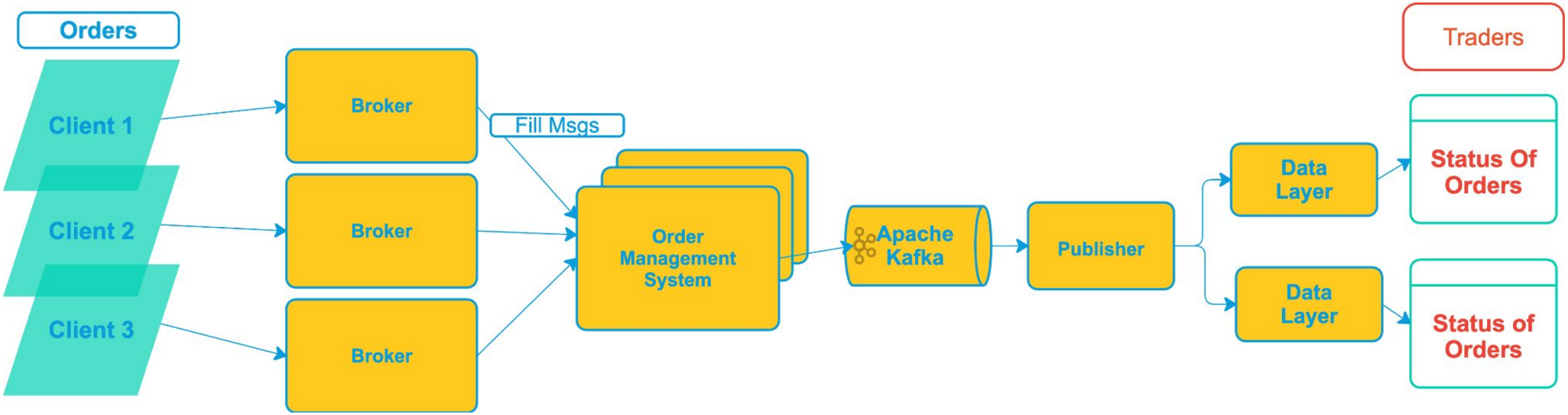


TechAtBloomberg.com

Bloomberg

Engineering

Equity Order Execution: An End-to-End Workflow



TechAtBloomberg.com

Bloomberg

Engineering

Why End-to-End Latency Monitoring Matters

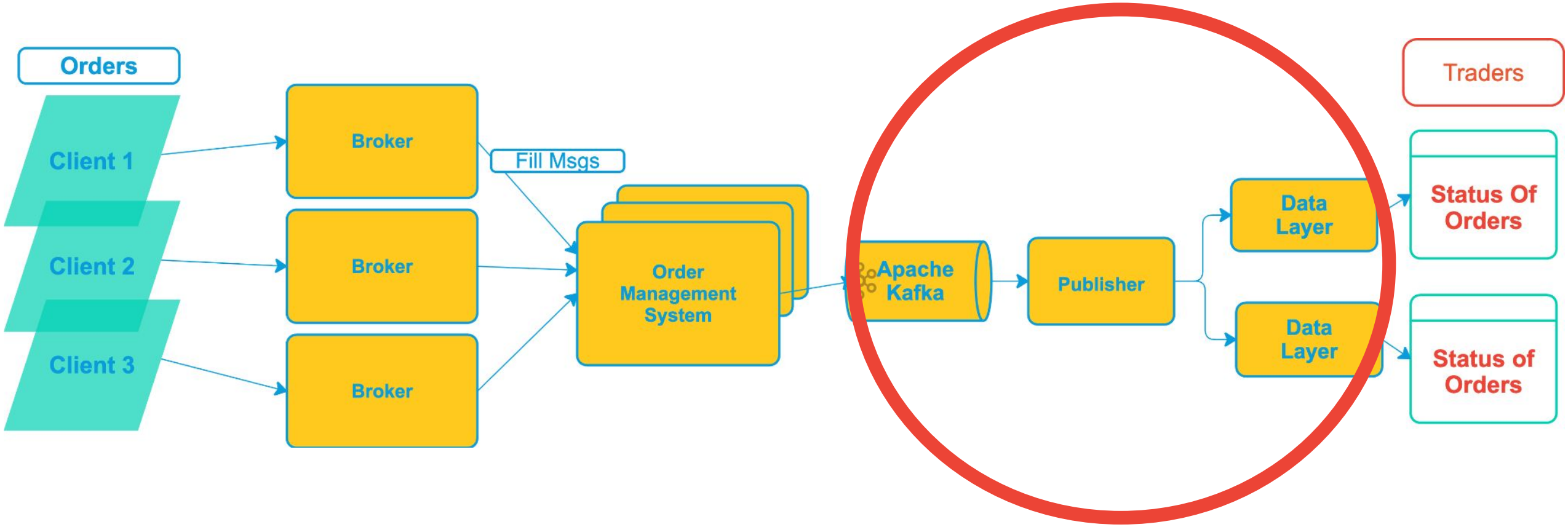
- Measure user experience
  - In complex, distributed architectures
- Alert on spikes
  - Improve time to resolution
- Service Level Objectives (SLOs)
  - Measurable targets for reliability and performance
  - "99.9% of requests should finish within 1s"
  - For engineers, managers, executives
- Design & development

TechAtBloomberg.com

Bloomberg

Engineering

Equity Order Execution: *Zooming In …*

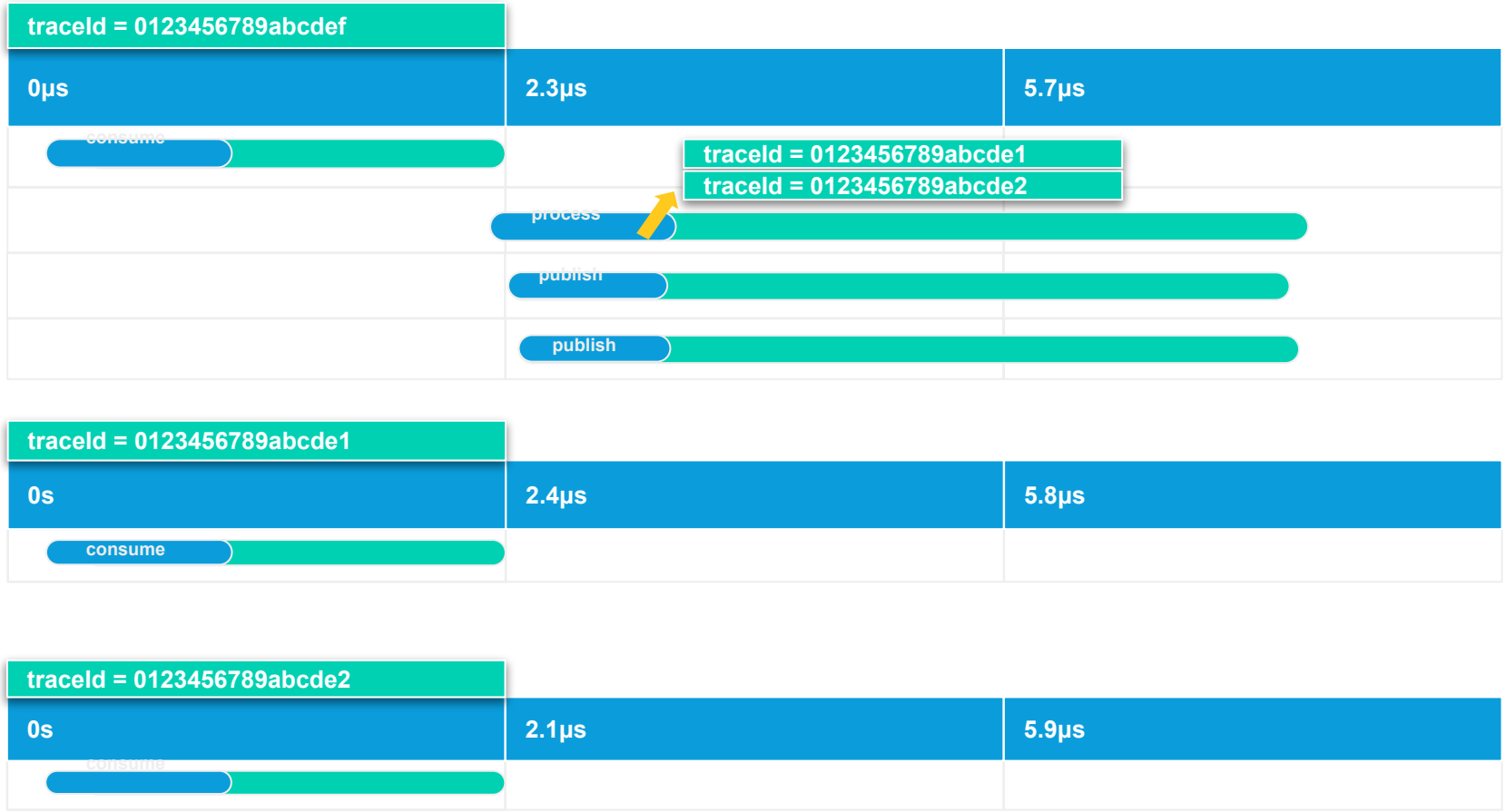


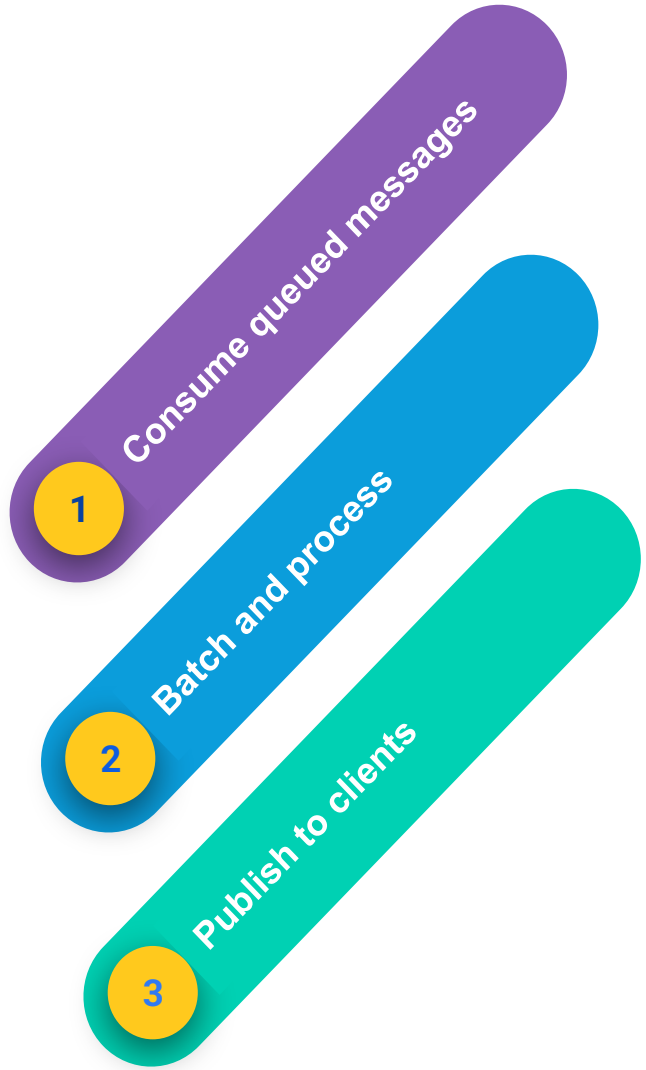
TechAtBloomberg.com

Bloomberg

Engineering

Example Distributed Trace



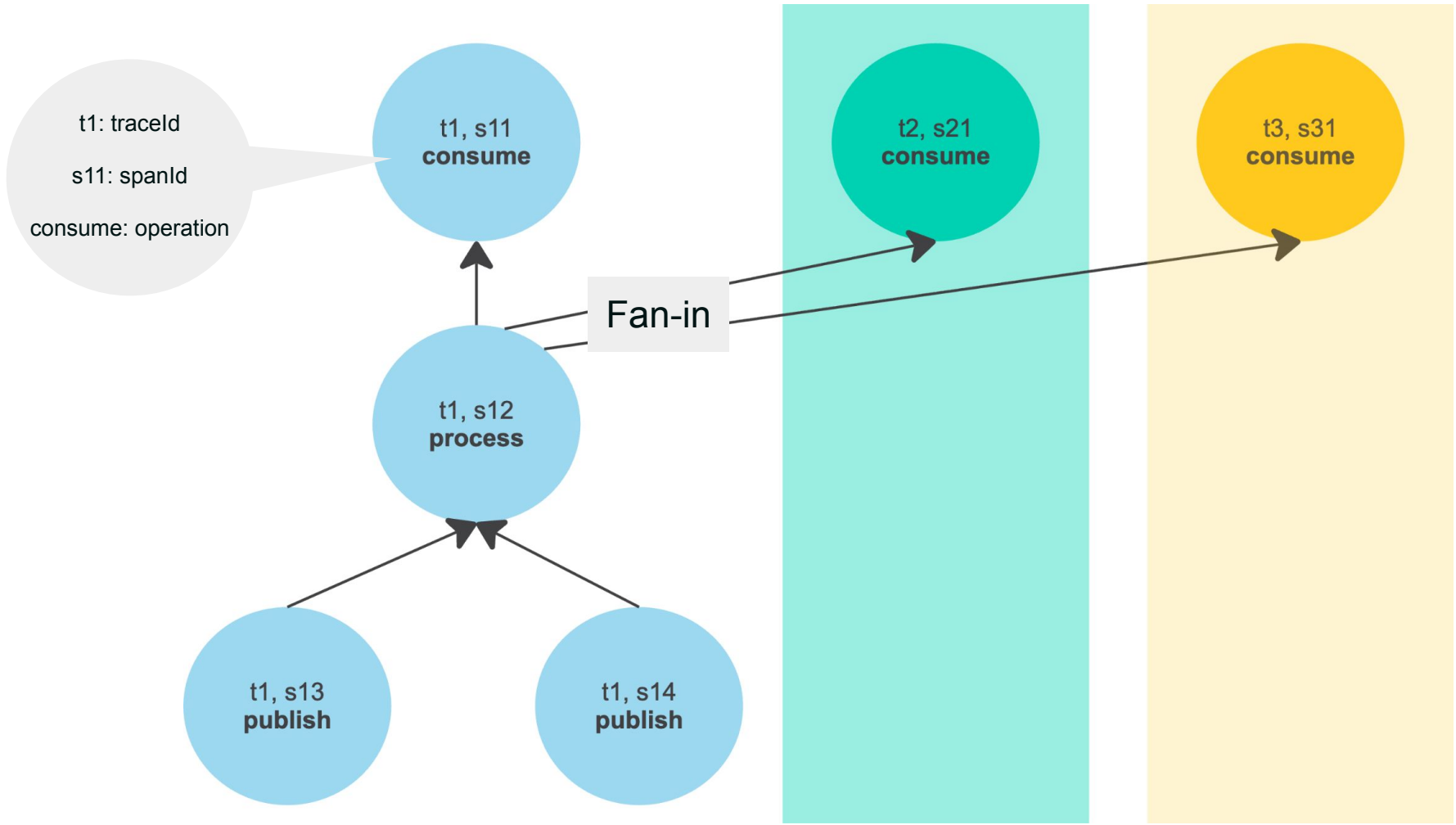


TechAtBloomberg.com

Bloomberg

Engineering

Trace = Directed Acyclic Graph (DAG)



TechAtBloomberg.com

Bloomberg

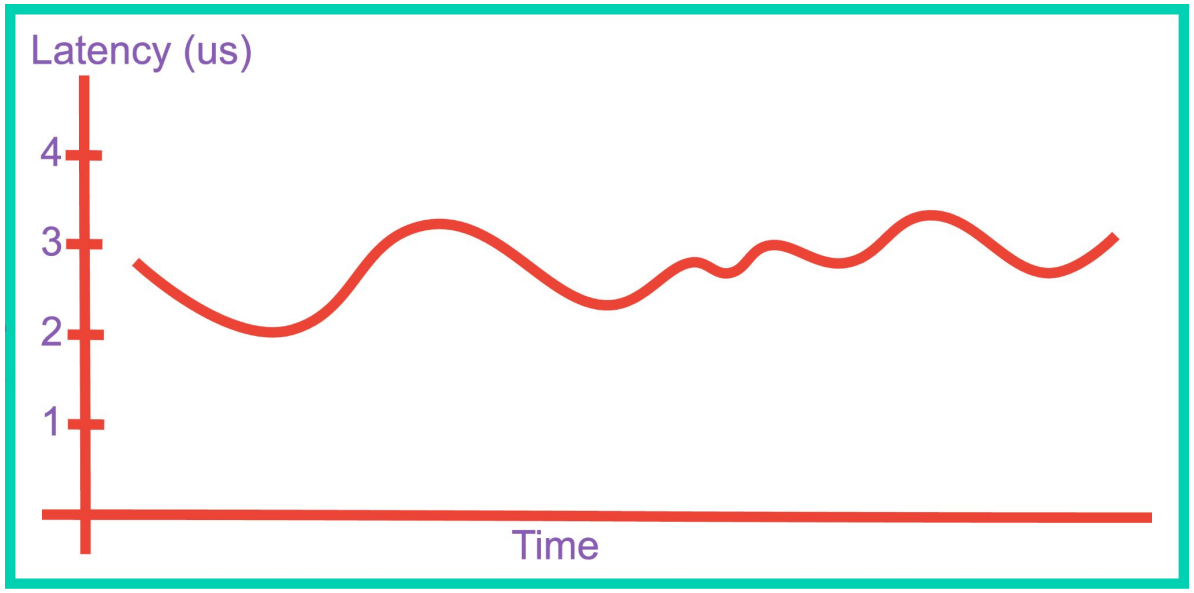
Engineering

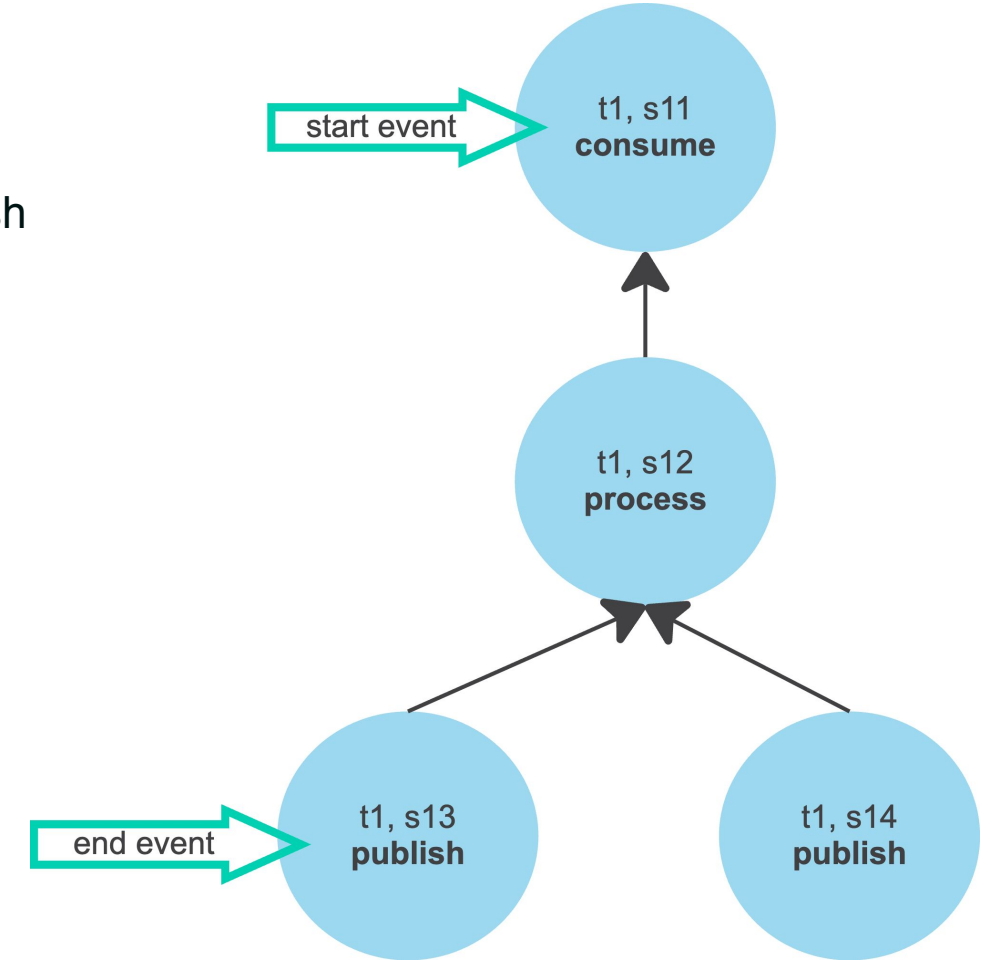
Rule = Defines End-to-End Workflow

*Get me the latency of requests going from **consume** to **publish***

Rule = consumeToPublish

start = consume, **end** = publish



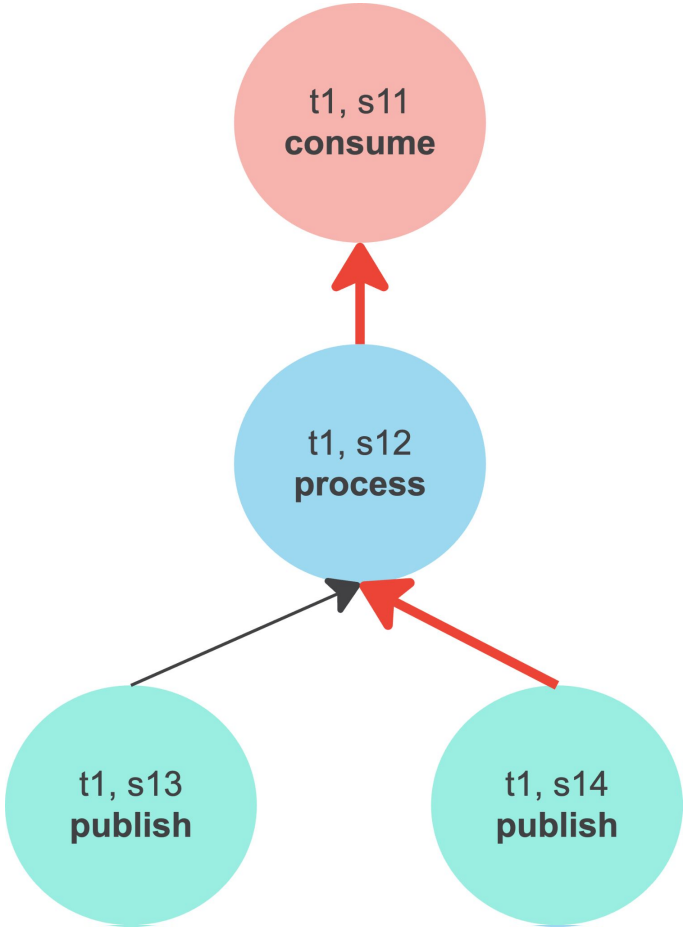


TechAtBloomberg.com

Bloomberg

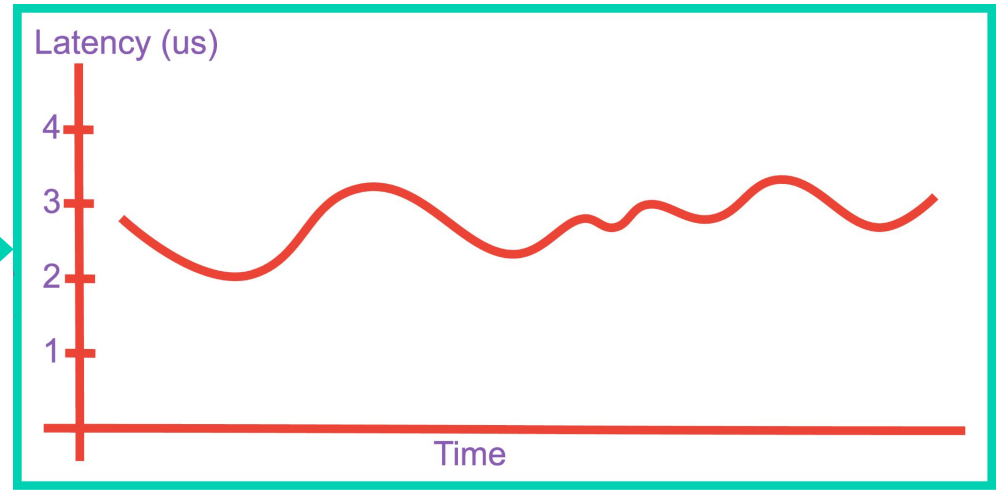
Engineering

End-to-End Latency From *One* Trace



Difference between
start of start event
&
end of end event

Path 1: **3µs**
Path 2: **3.5µs**

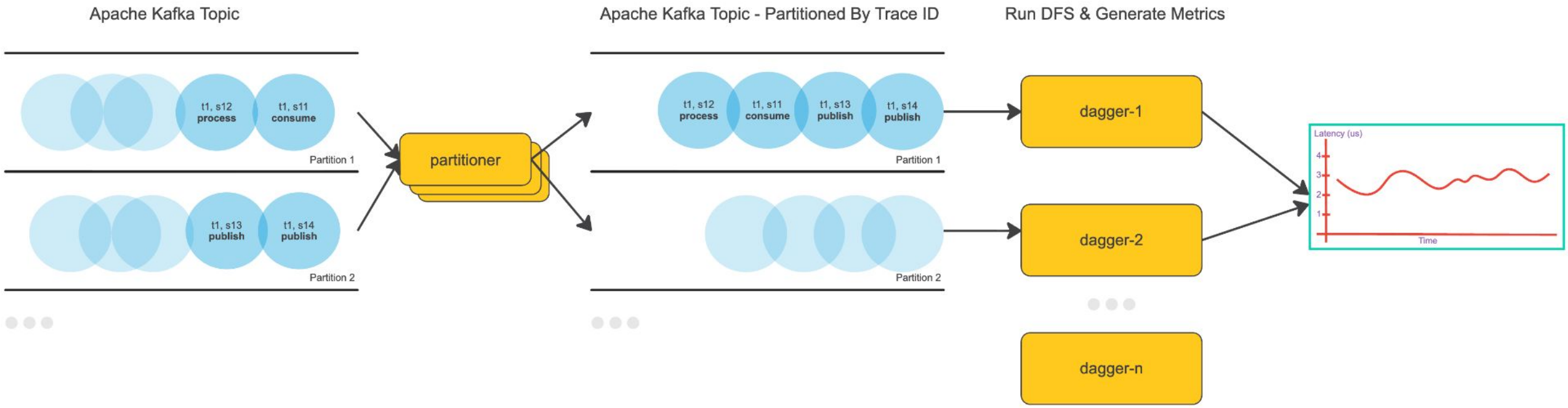


Depth-First Search (DFS) through DAG

TechAtBloomberg.com

Bloomberg
Engineering

Handling Traces At Scale

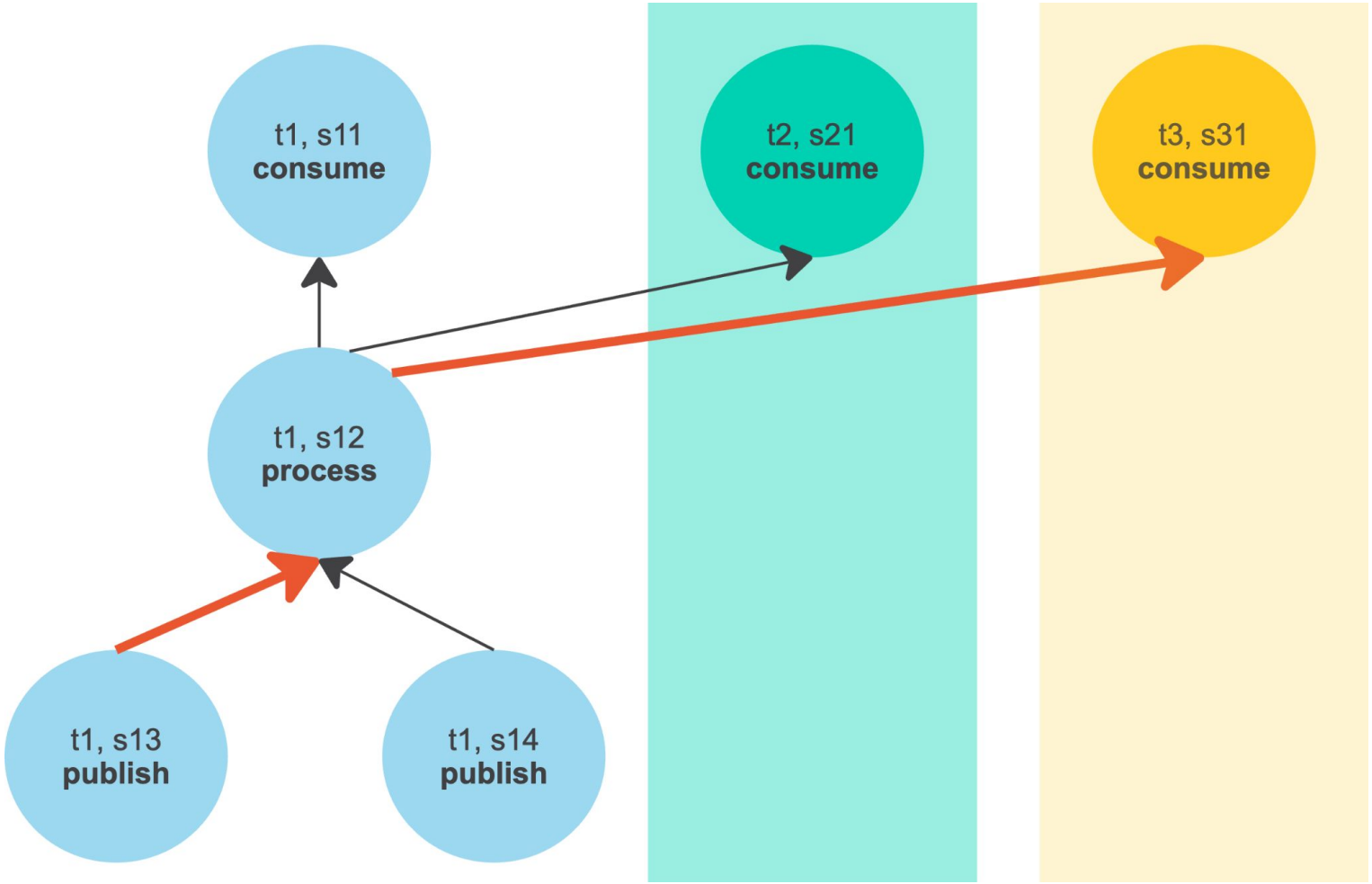


TechAtBloomberg.com

Bloomberg

Engineering

Enter … *(drumroll please)* … Fan-ins!



Problem
Scaling with fan-ins

All spans from the same *weakly connected DAG* should be handled together.

Let's call this a trace "***bundle.***"

TechAtBloomberg.com

Bloomberg

Engineering

Two Approaches, One Thing In Common - C++

Memory Control

Concurrency At Scale

Performant Standard Data Structures

Mature Tooling & Ecosystem

TechAtBloomberg.com

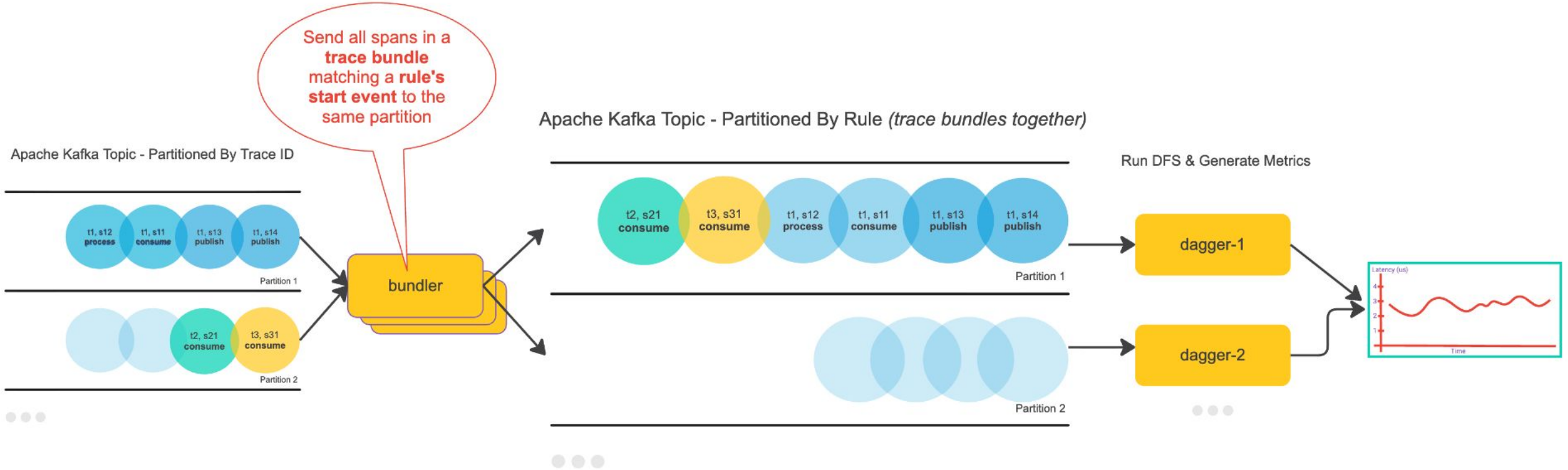
Bloomberg

Engineering

Agenda: Digging Into the Two Approaches

- Designs for the approaches
  - Interaction between different C++ microservices
- C++ data structures used
- Pinpoint implementation-level bottlenecks
  - Use of profiling for improving performance
- Summarize pros and cons
- Key takeaways from this journey

Approach #1: Aggregating Trace "Bundles" By Rule

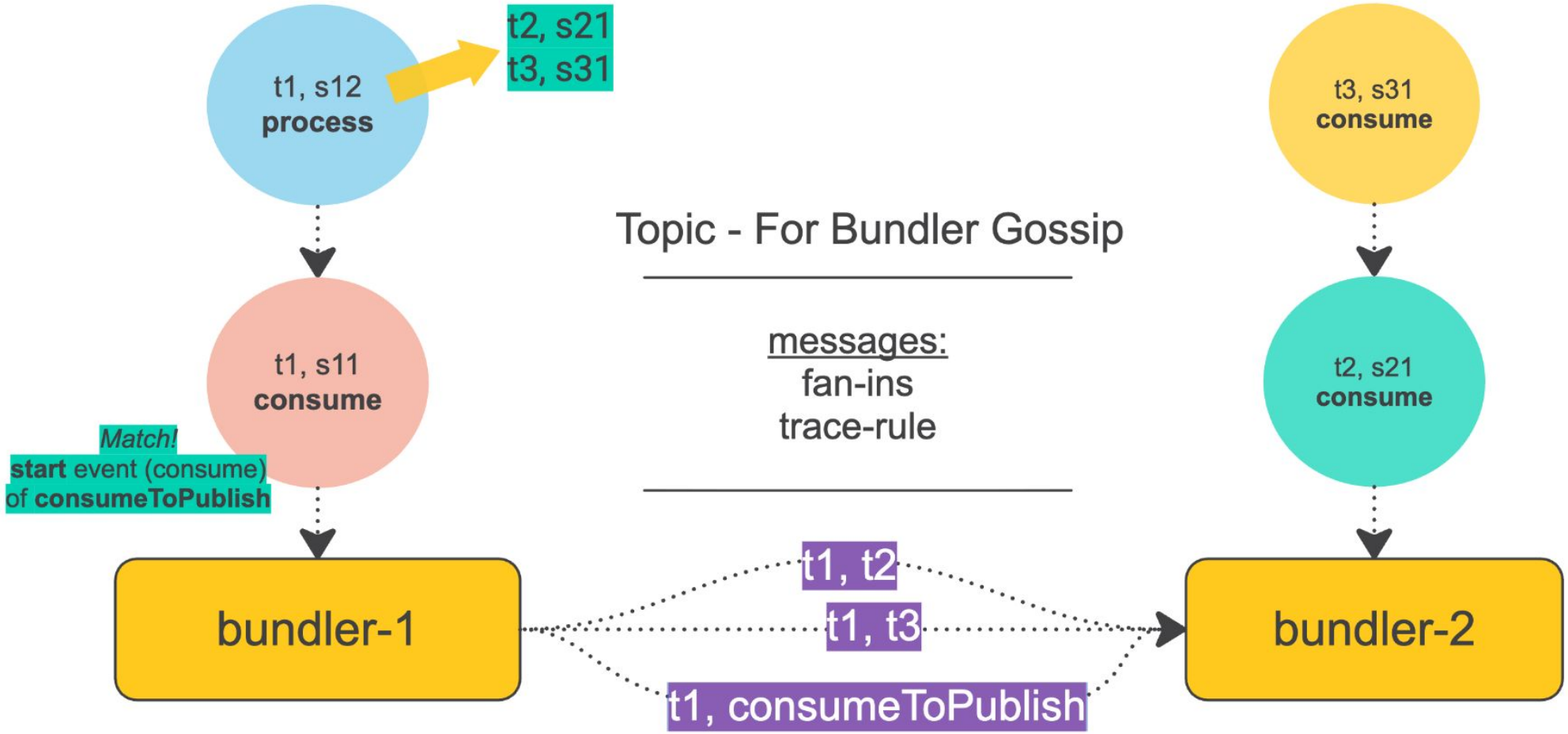


TechAtBloomberg.com

Bloomberg

Engineering

Bundler for Merging Traces Bundles

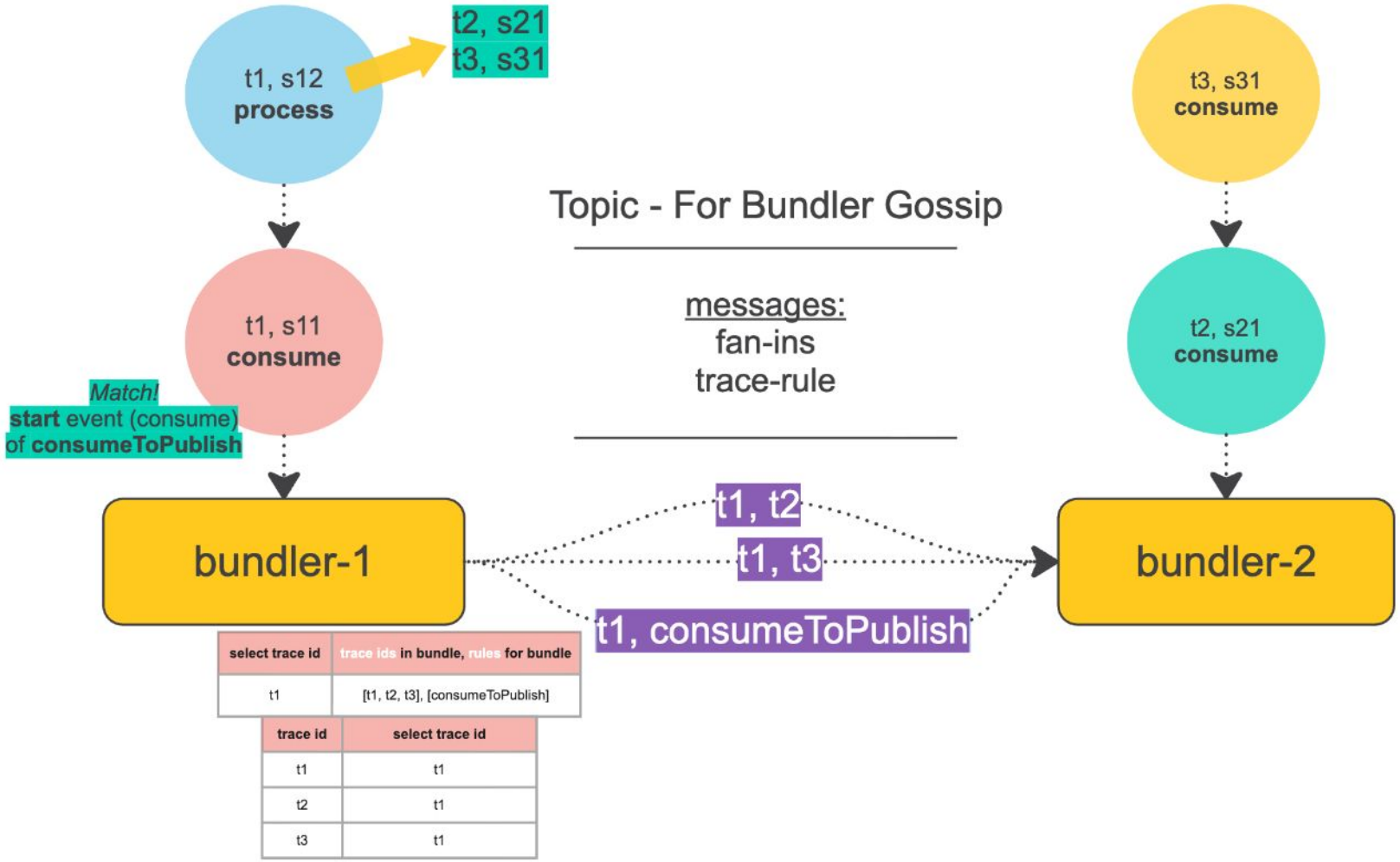


TechAtBloomberg.com

Bloomberg

Engineering

Bundler's Data Structures



In-memory maps

select trace id	trace ids in bundle, rules for bundle
t1	[t1, t2, t3], [consumeToPublish]

trace id	select trace id
t1	t1
t2	t1
t3	t1

TechAtBloomberg.com

Bloomberg

Engineering

Zooming Into Bundler

```
unordered_map<TraceId, TraceId>          traceToBundleKey;

struct TracesRules {
    unordered_set<string>                rules;
    unordered_set<TraceId>               tracesInBundle;
};

unordered_map<TraceId, TracesRules> bundleKeyToTracesRules;
```

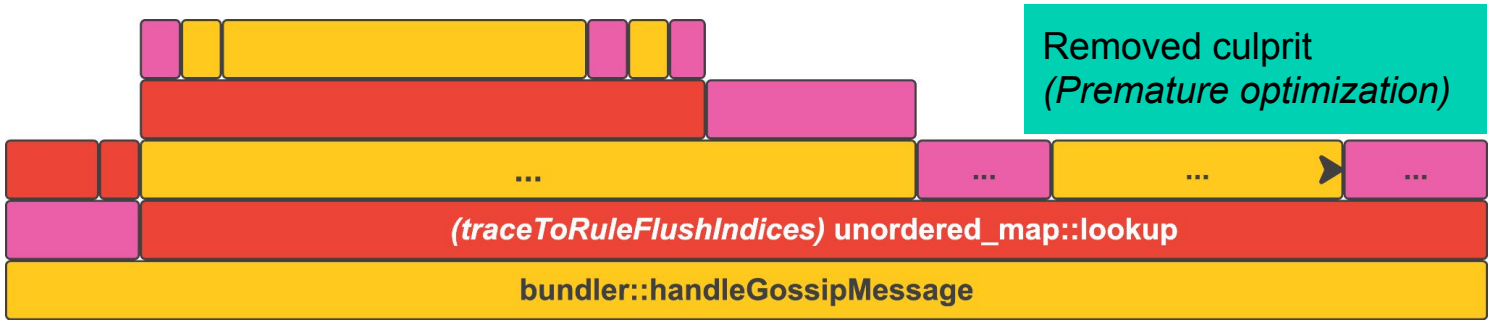
TechAtBloomberg.com

Bloomberg

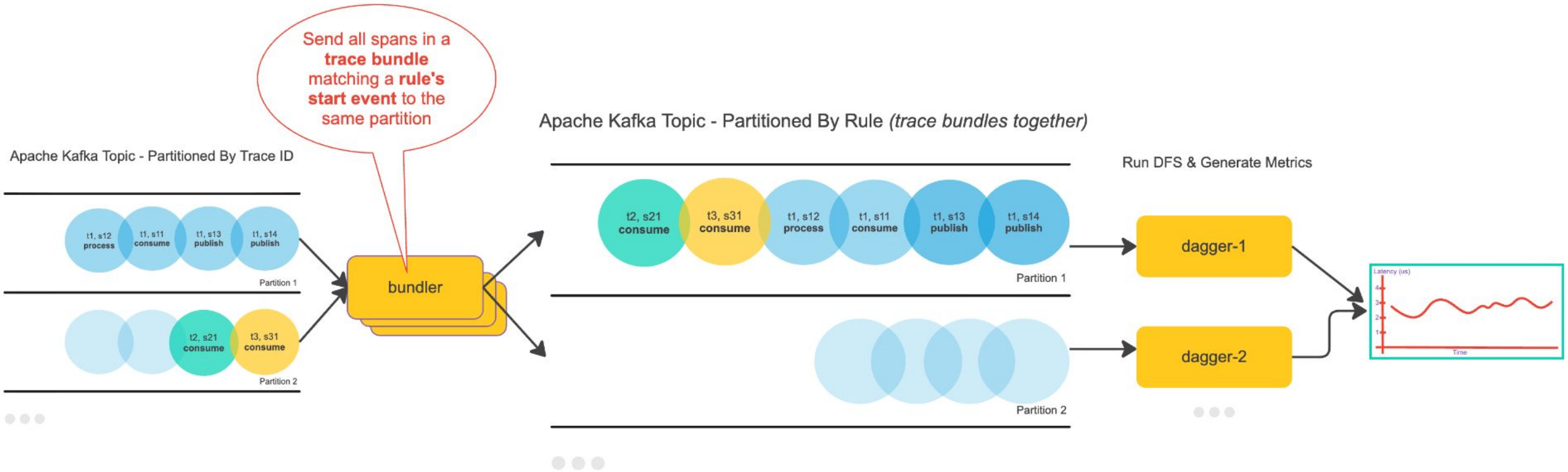
Engineering

But, It Wasn't Always This Way!

- Handling the messages between *bundler*'s
  - Slow: lag in Kafka consumption
- Previously, we had
  - unordered_map<TraceId, TraceId> traceToBundleKey;
  - unordered_map<TraceId, unordered_set<TraceId>> bundleKeyToTraces;
  - unordered_map<TraceId, unordered_map<string, int>> **traceToRuleFlushIndices**;
- Profiled handler
  - Culprit: Lookups/writes in **traceToRuleFlushIndices**



Review: Aggregating Trace "Bundles" By Rule



TechAtBloomberg.com

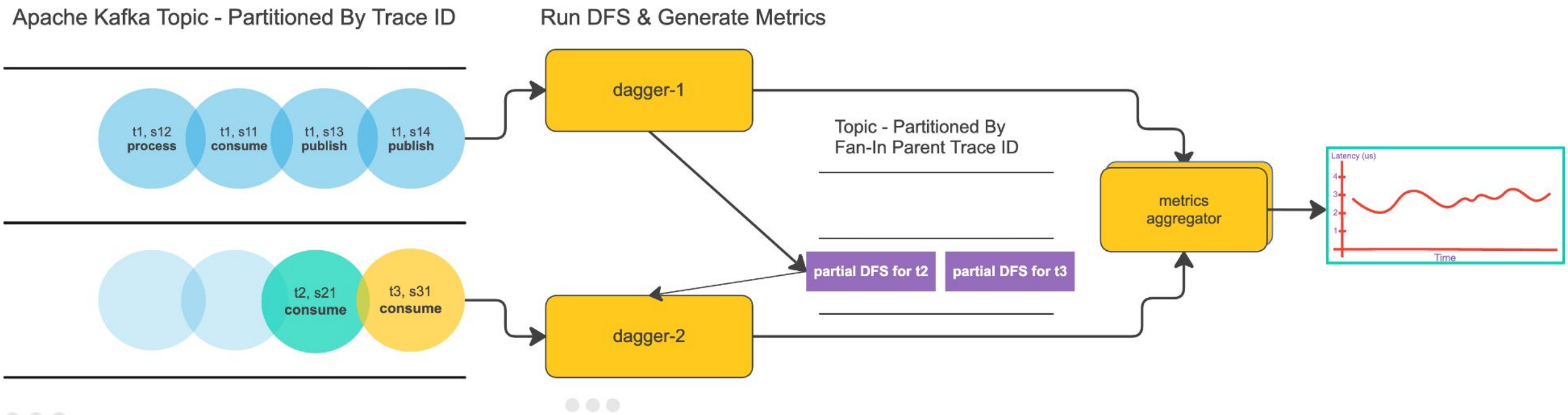
Bloomberg

Engineering

Summarizing *Approach #1*

- Puts all spans from a trace bundle in the same partition
- Works well when rules match a small portion of traces
- Constraints:
  - Load imbalance
    - Rule with high volume trace matches bloat *dagger* memory
    - e.g., rules on middleware spans
  - Fan-in memory bloat
    - Fan-in traces not matching any rules exacerbate imbalance

Approach #2: Handing Off "Partial-DFS"

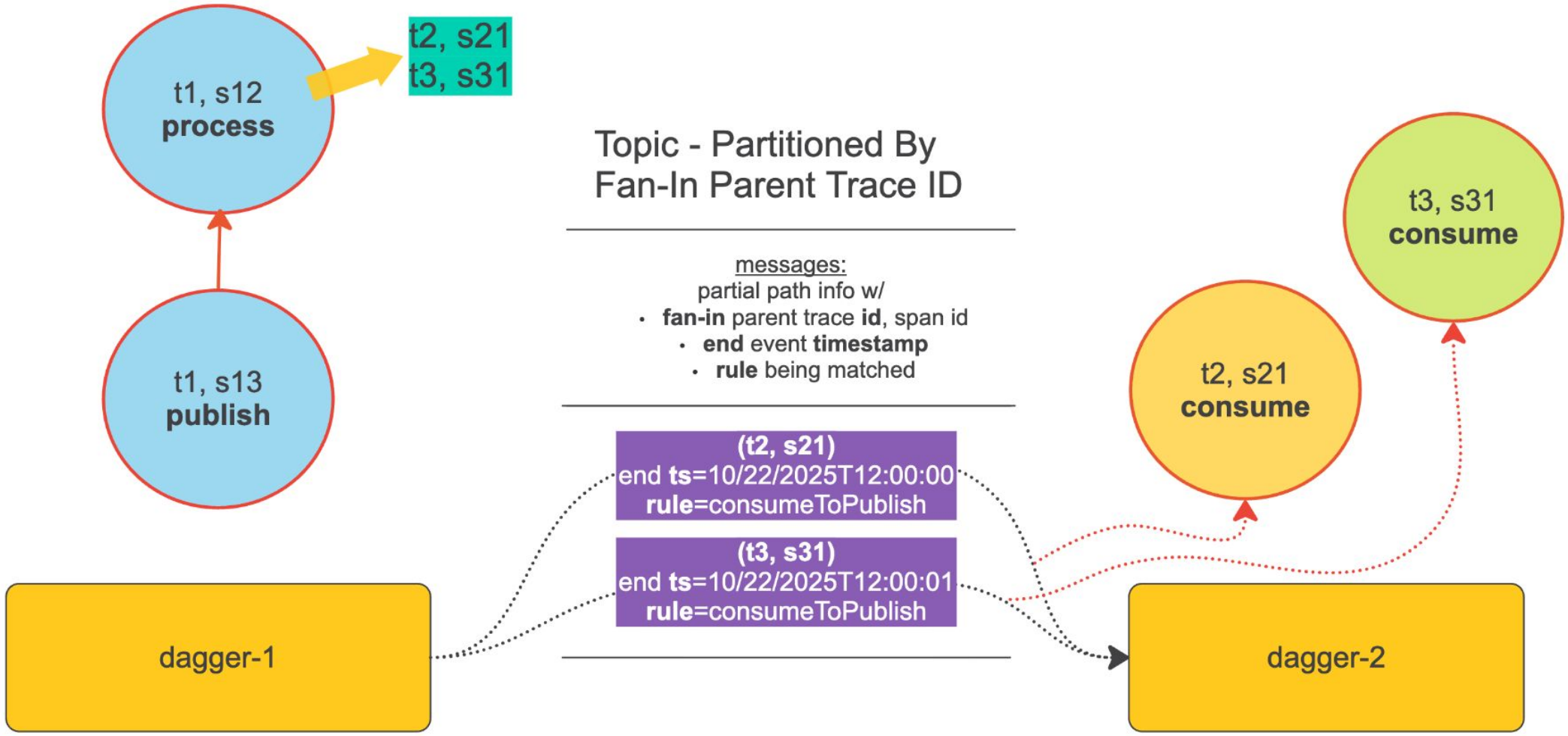


TechAtBloomberg.com

Bloomberg

Engineering

Gossip About Partial-DFS



TechAtBloomberg.com

Bloomberg

Engineering

Zooming Into Dagger

```
struct TraceData {
    unordered_map<SpanId, Span>                    spans;
    vector<string>                                 rules;
};

unordered_map<TraceId, shared_ptr<TraceData>>    tracesToProcess;

unordered_map<TraceId, FanInData>                partialDfsToProcess;
```

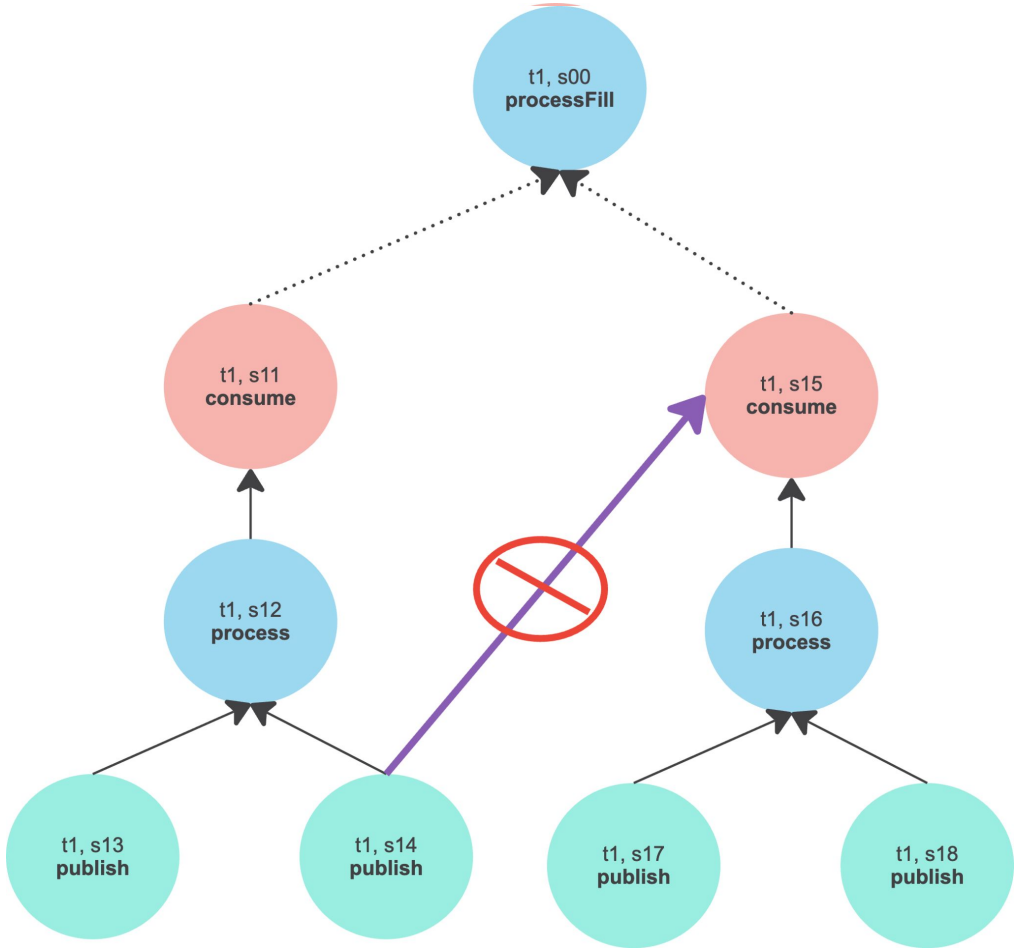
TechAtBloomberg.com

Bloomberg

Engineering

But, Processing Some Traces Took Long!

- Running the DFS over a specific trace
  - Processing time occasionally in several seconds
- Special user configuration: *skipDFS*
  - Start and end event in the same trace
  - NOT(multiple starts && multiple ends)



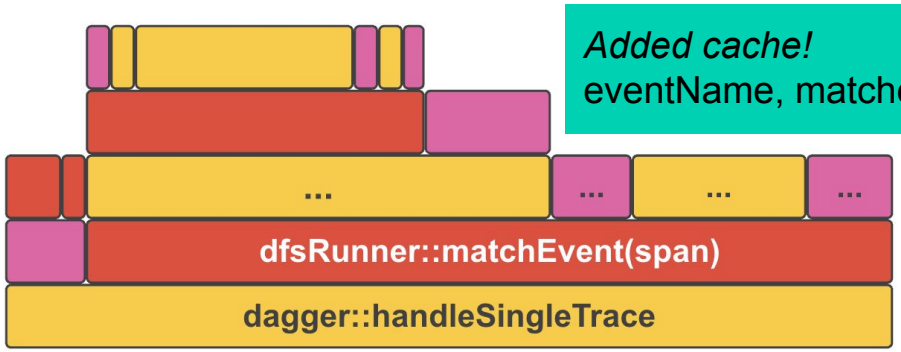
TechAtBloomberg.com

Bloomberg

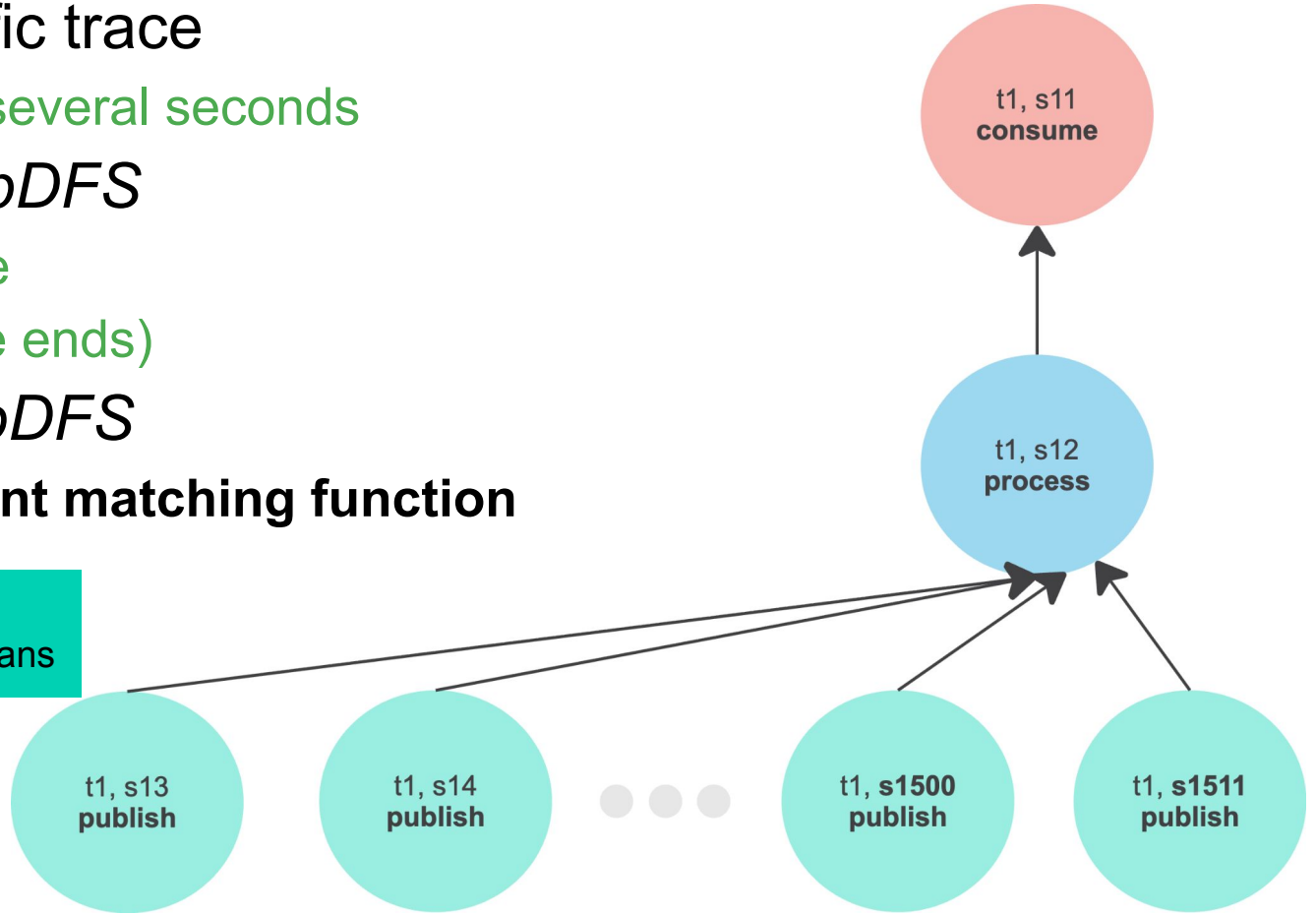
Engineering

But, Processing Some Traces Took Long!

- Running the DFS over a specific trace
  - Processing time occasionally in several seconds
- Special user configuration: *skipDFS*
  - Start/end event in the same trace
  - NOT(multiple starts AND multiple ends)
- Profiled DFS execution for *skipDFS*
  - Culprit: repetitive calls to the **event matching function**



Added cache!
eventName, matchedSpans



TechAtBloomberg.com

Bloomberg

Engineering

Summarizing *Approach #2*

- Lazy approach (as opposed to #1, greedy)
- Send info regarding fan-ins only when relevant to DFS
- Avoids load imbalance
- Constraints:
  - Deep, chained fan-ins
    - Consecutive fan-ins in one path incur gossip delays

Conclusion

End-to-end latency metrics from complex traces can be generated at scale.

- Pick the best approach for you
- Design for speed from the start
  - Avoid premature optimizations
  - Leverage tooling to iteratively improve code performance
- End-to-end latency monitoring is crucial
  - Trace provides a unified solution
  - Client impact can be directly monitored
- Distributed, streaming DAGs are fun!

TechAtBloomberg.com

Bloomberg

Engineering

Thank you!

Bloomberg

Engineering

Let's connect:

kmaharshi@bloomberg.net
www.linkedin.com/in/kusha-maharshi

Learn more: www.TechAtBloomberg.com/cplusplus

We are hiring: bloomberg.com/engineering

TechAtBloomberg.com

Using Distributed Trace for End-to-End Latency Metrics

Bloomberg

Engineering

CppCon 2025
September 18, 2025

Kusha Maharshi
Senior Software Engineer, Telemetry Infrastructure

TechAtBloomberg.com