Tail Latency in Node.js:

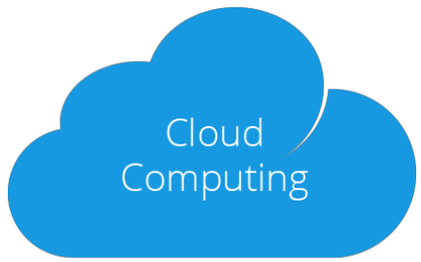
Energy Efficient Turbo Boosting for Long Tail Requests in JavaScript

Wenzhi Cui
Daniel Richins
Yuhao Zhu
Vijay Janapa Reddi

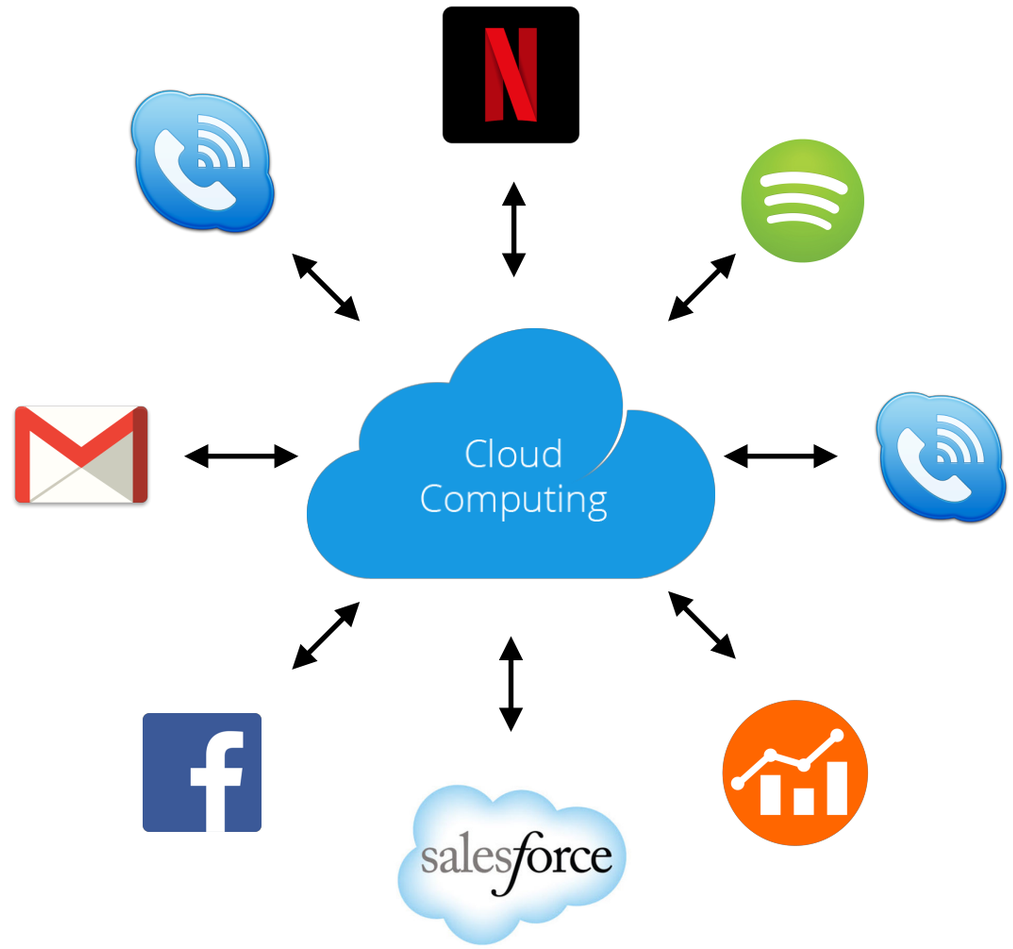
UNIVERSITY of ROCHESTER

Google

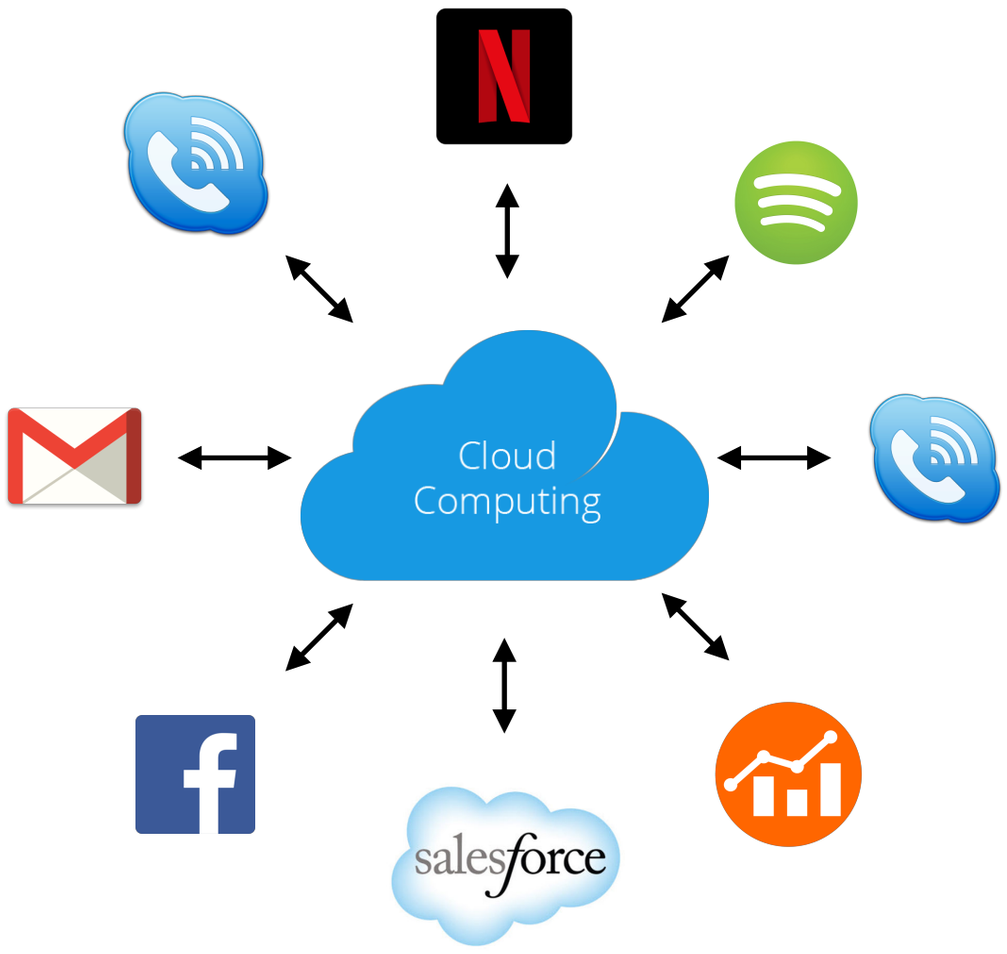
HARVARD UNIVERSITY


Cloud
Computing

Connecting People (2010s)



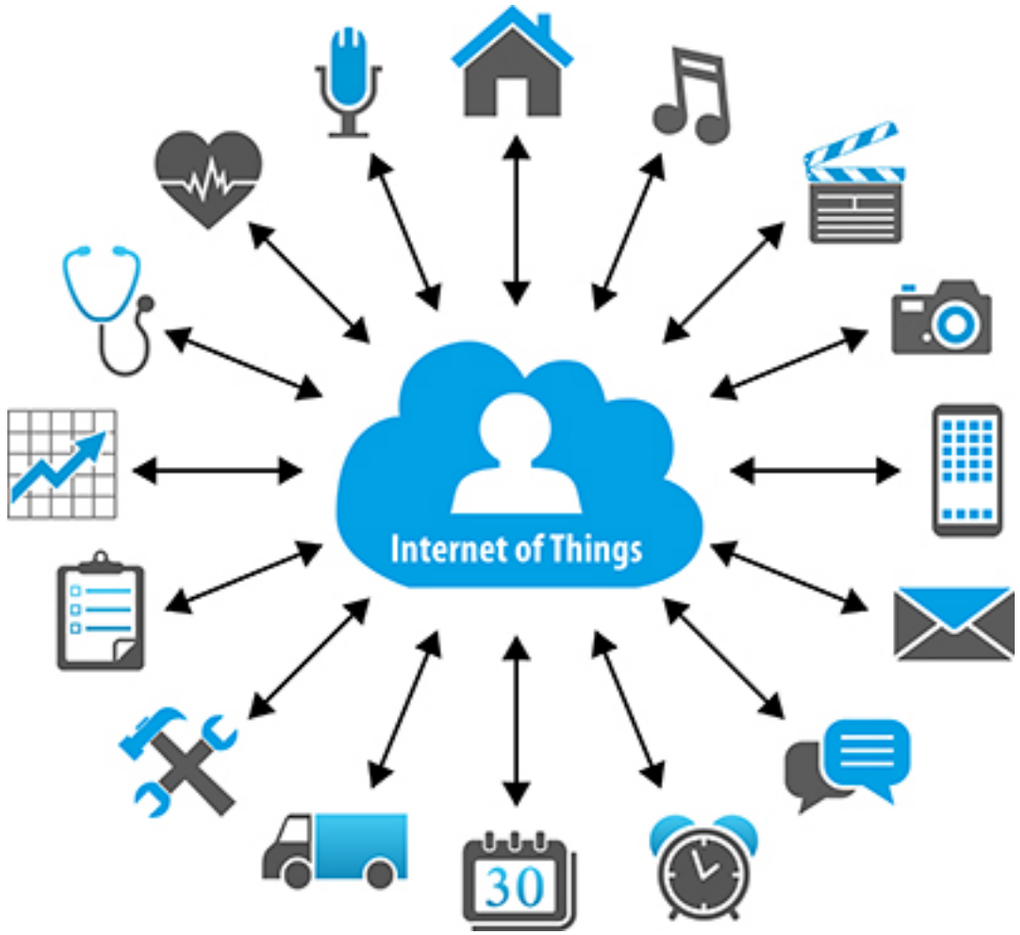
Connecting People (2010s)



1 billion
Gmail users

>1 billion
users

Connecting Things (2020s)



Connecting Things (2020s)

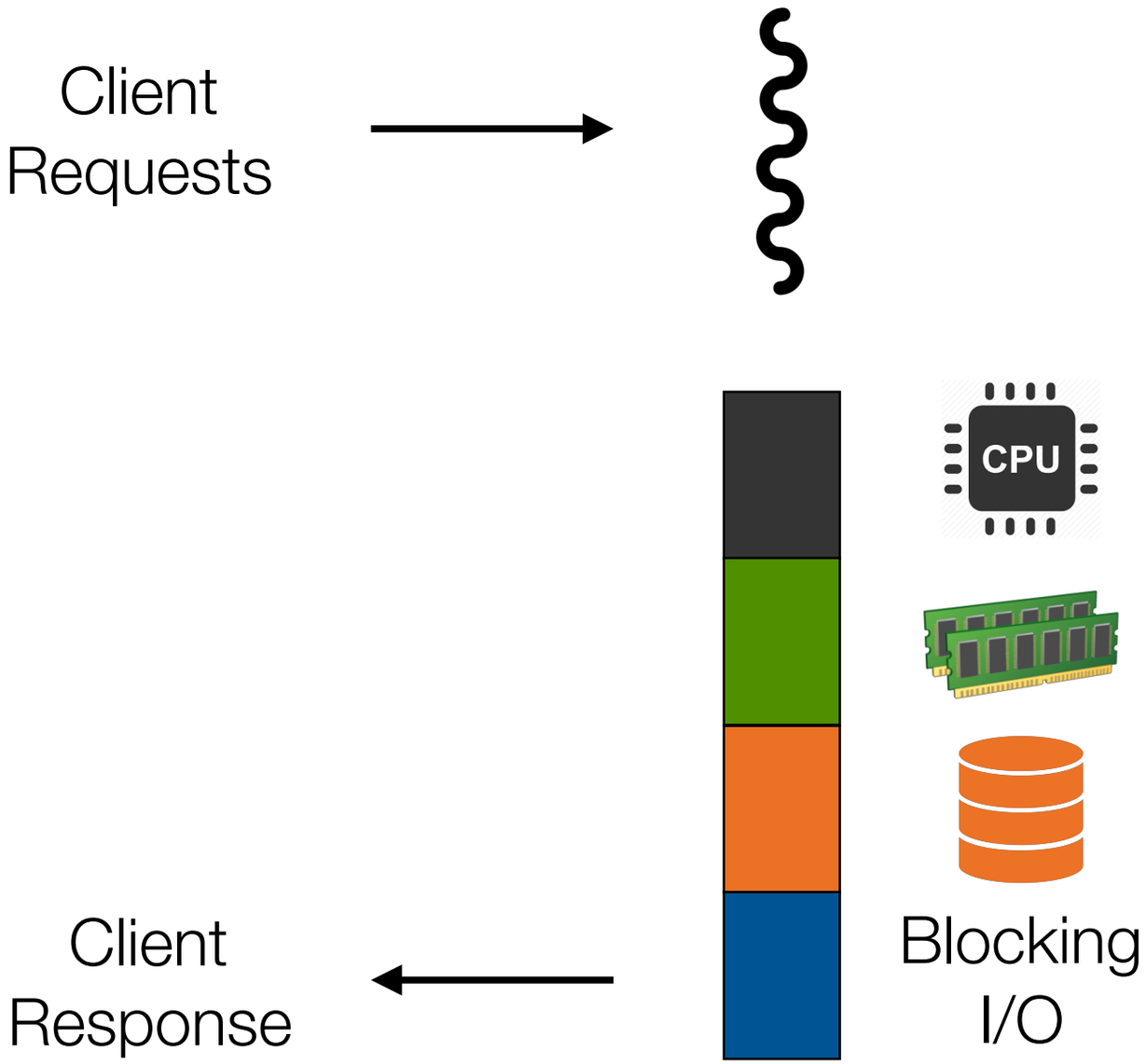
50 Billion Devices

"The Internet of Things" — Cisco
www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

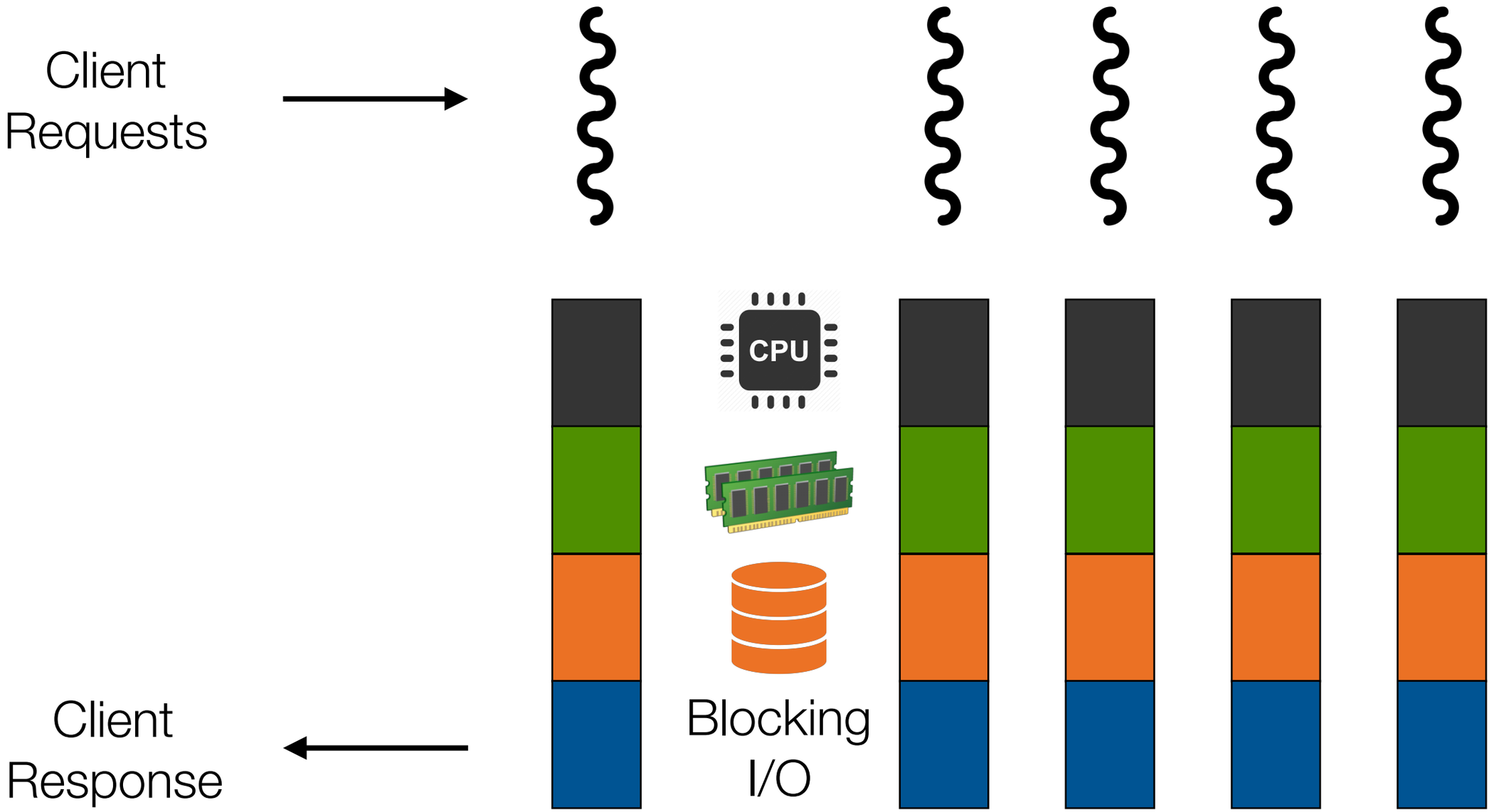
Thread-based Programming: Traditional Approach

Client Requests

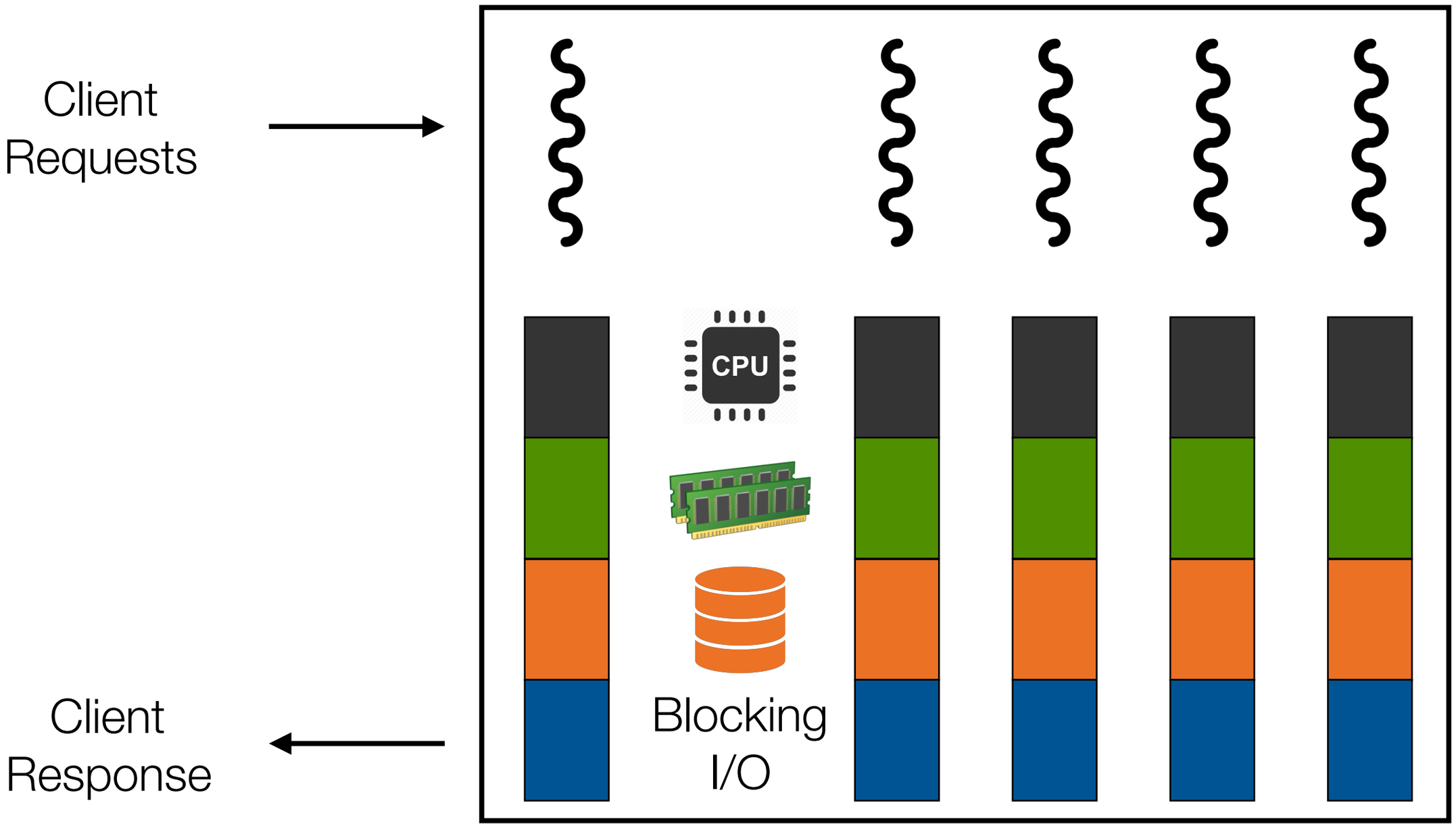
Thread-based Programming: Traditional Approach



Thread-based Programming:
Traditional Approach



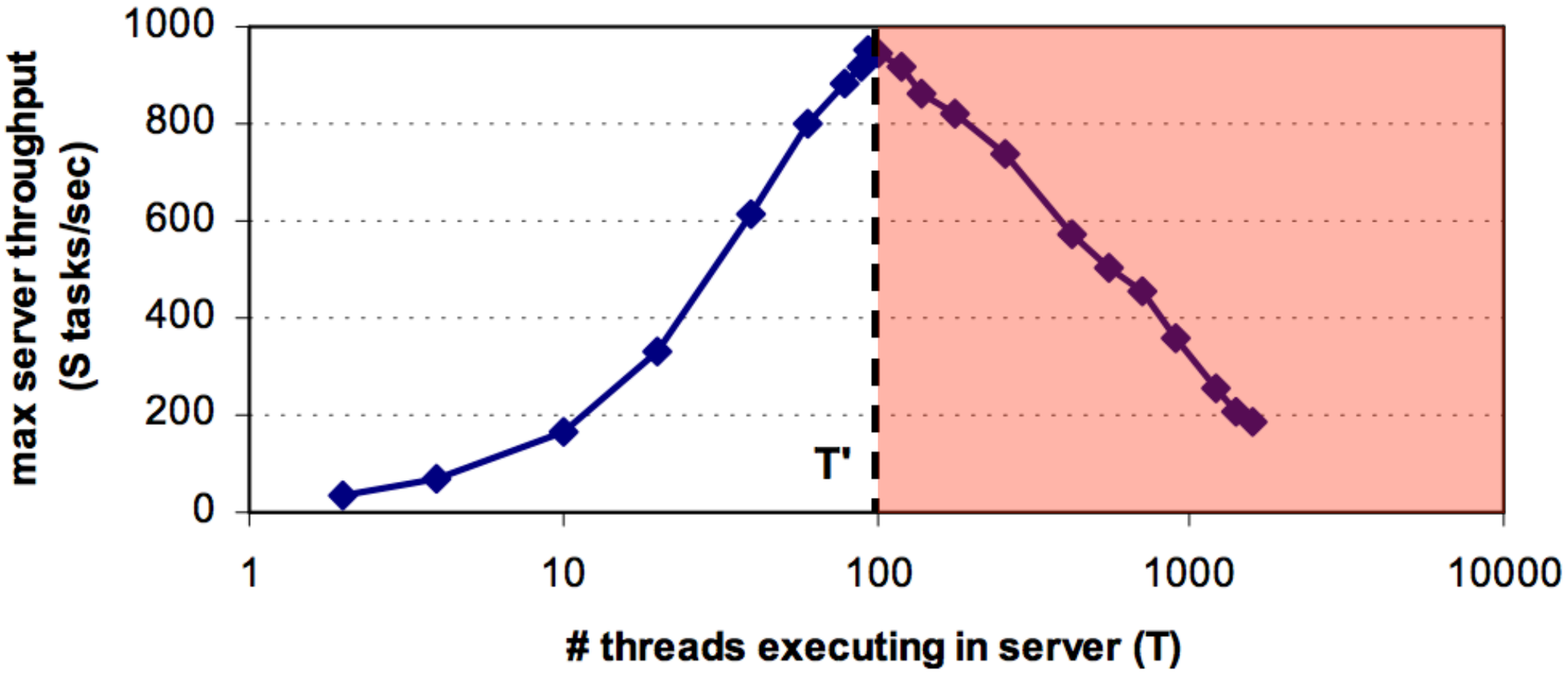
Thread-based Programming:
Traditional Approach



Limited resources & thrashing

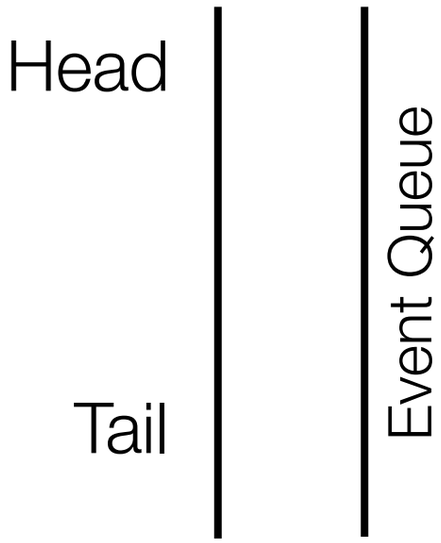
Thread-based Programming

[Welsh et al. ’00]

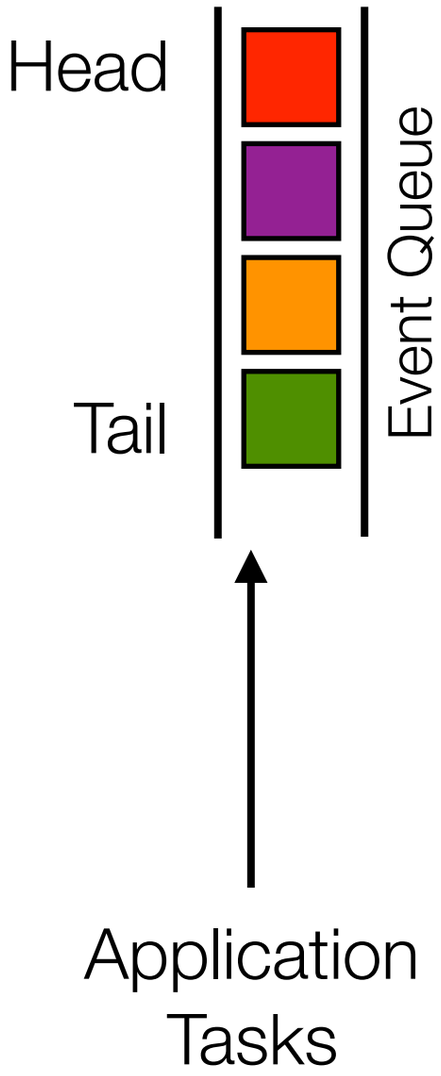


Event-driven Programming: Emerging Approach

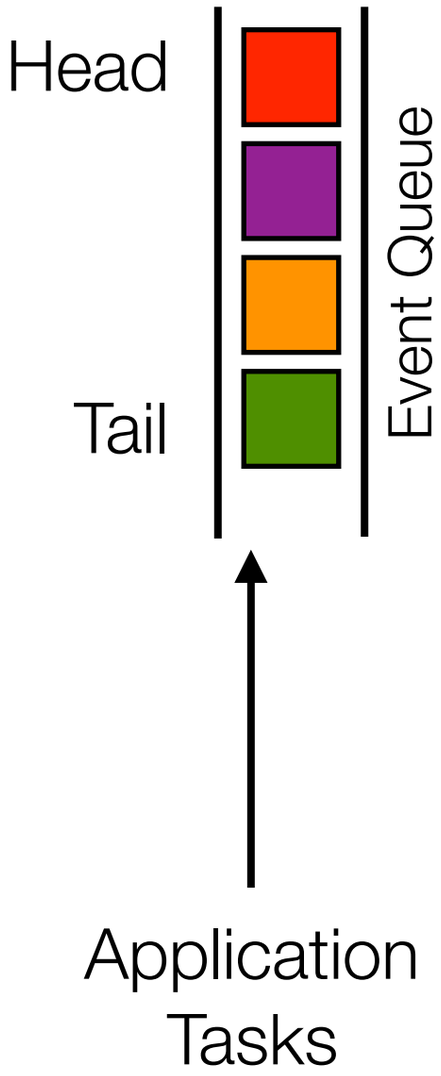
Event-driven Programming:
Emerging Approach



Event-driven Programming:
Emerging Approach

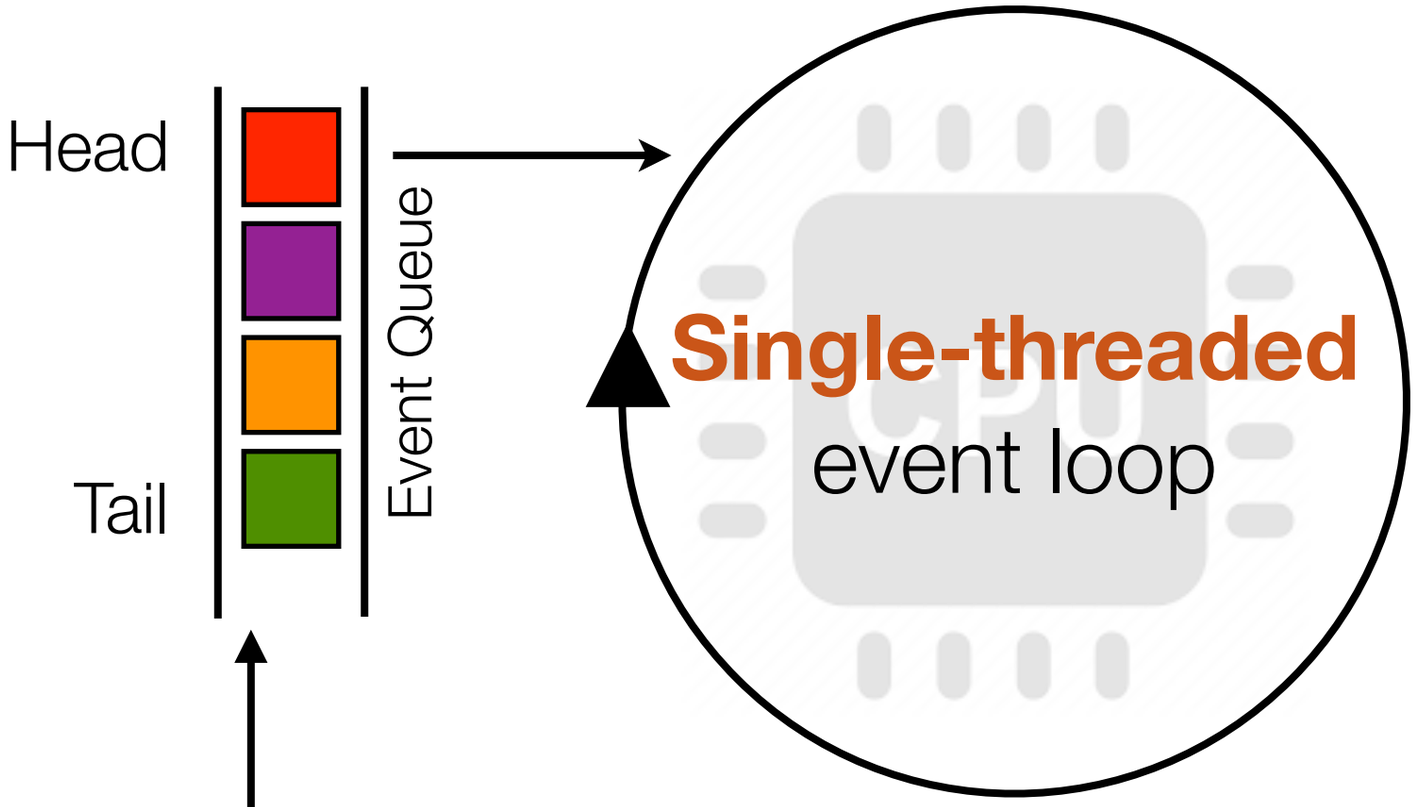


Event-driven Programming:
Emerging Approach



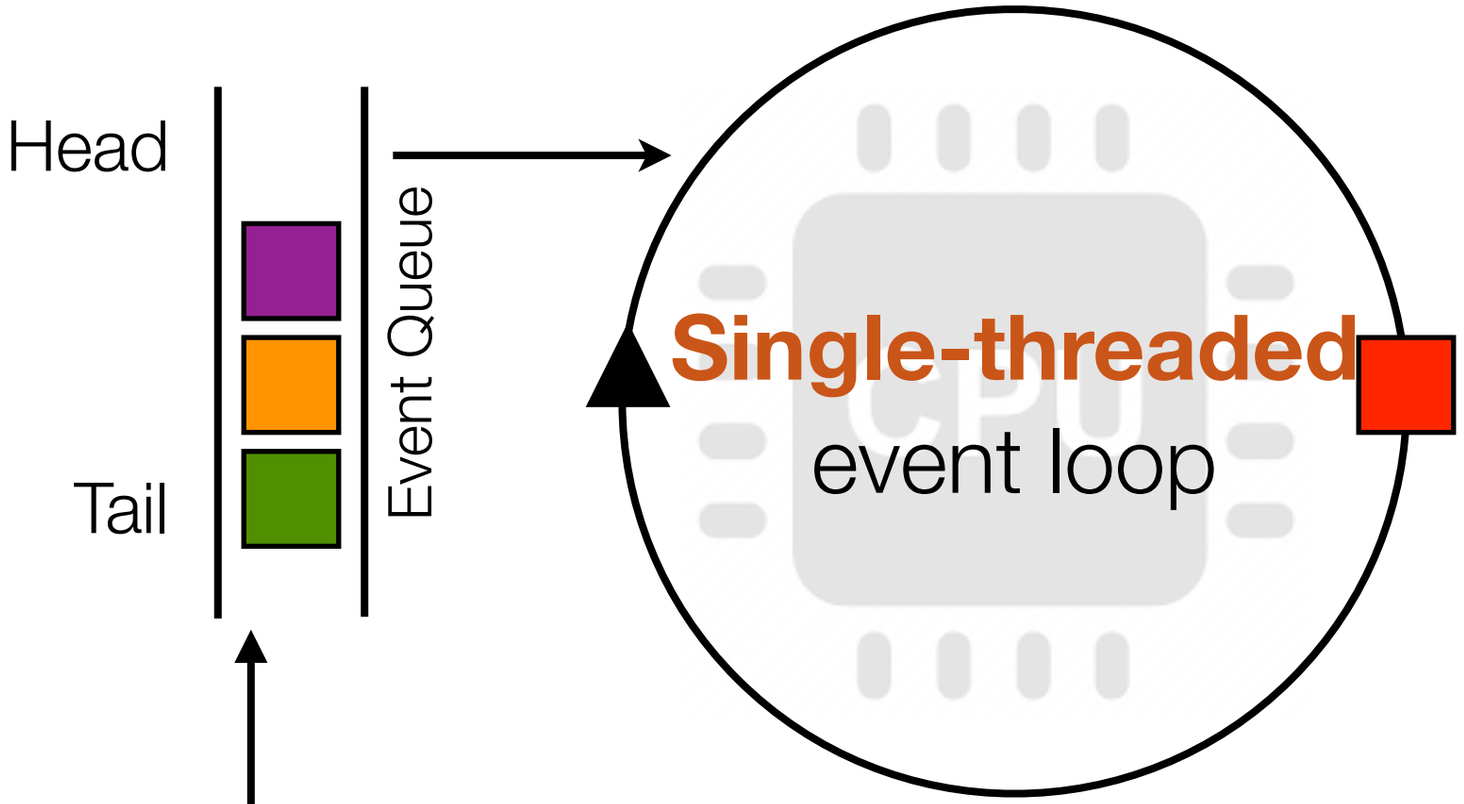

```
fs.readFile('input.txt',
  function (err, data) {
      if (err) return console.error(err);
      console.log(data.toString());
  }
);

console.log("Program continues…");
```

Event-driven Programming:
Emerging Approach



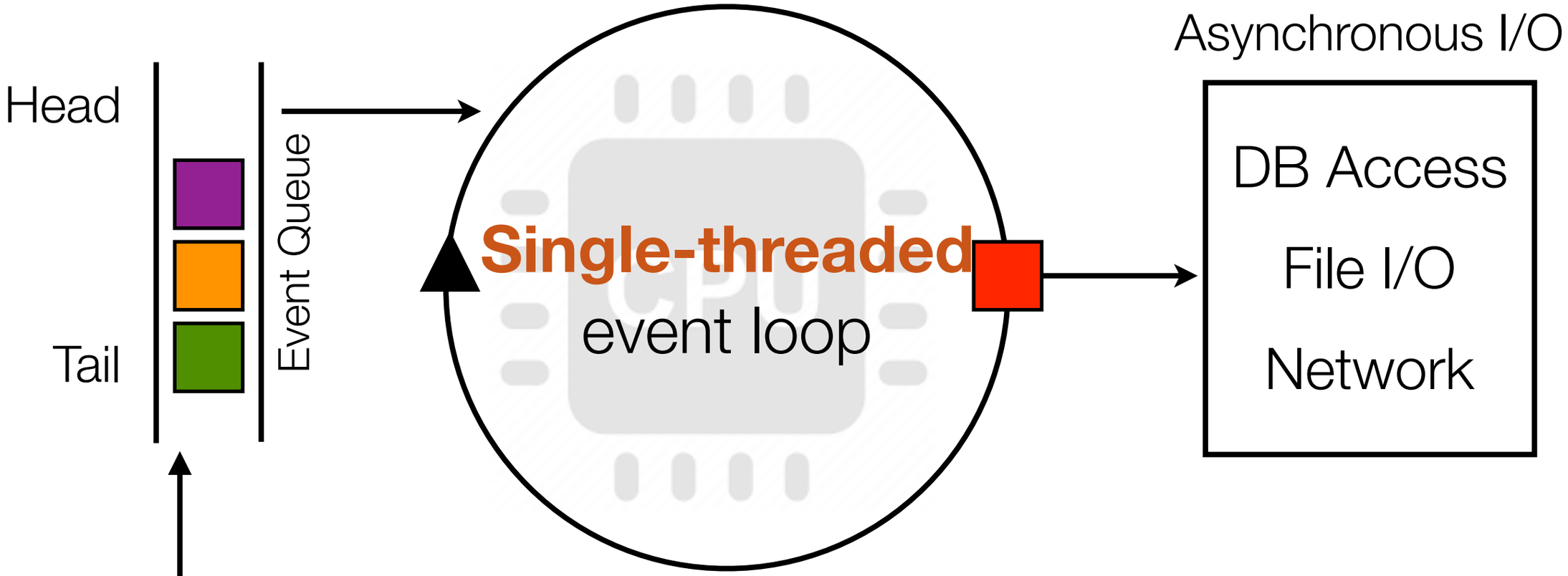

```
fs.readFile('input.txt',
  function (err, data) {
      if (err) return console.error(err);
      console.log(data.toString());
  }
);

console.log("Program continues…");
```

Event-driven Programming:
Emerging Approach



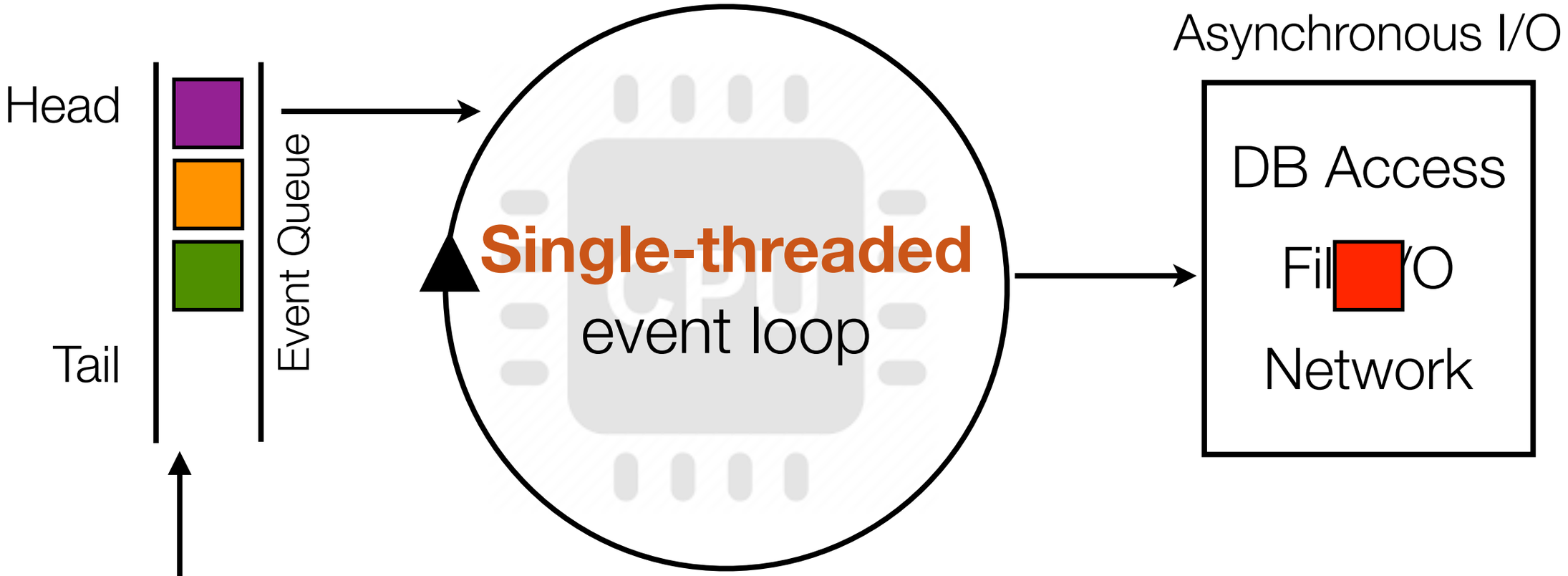

```
fs.readFile('input.txt',
  function (err, data) {
      if (err) return console.error(err);
      console.log(data.toString());
  }
);

console.log("Program continues…");
```

Event-driven Programming:
Emerging Approach



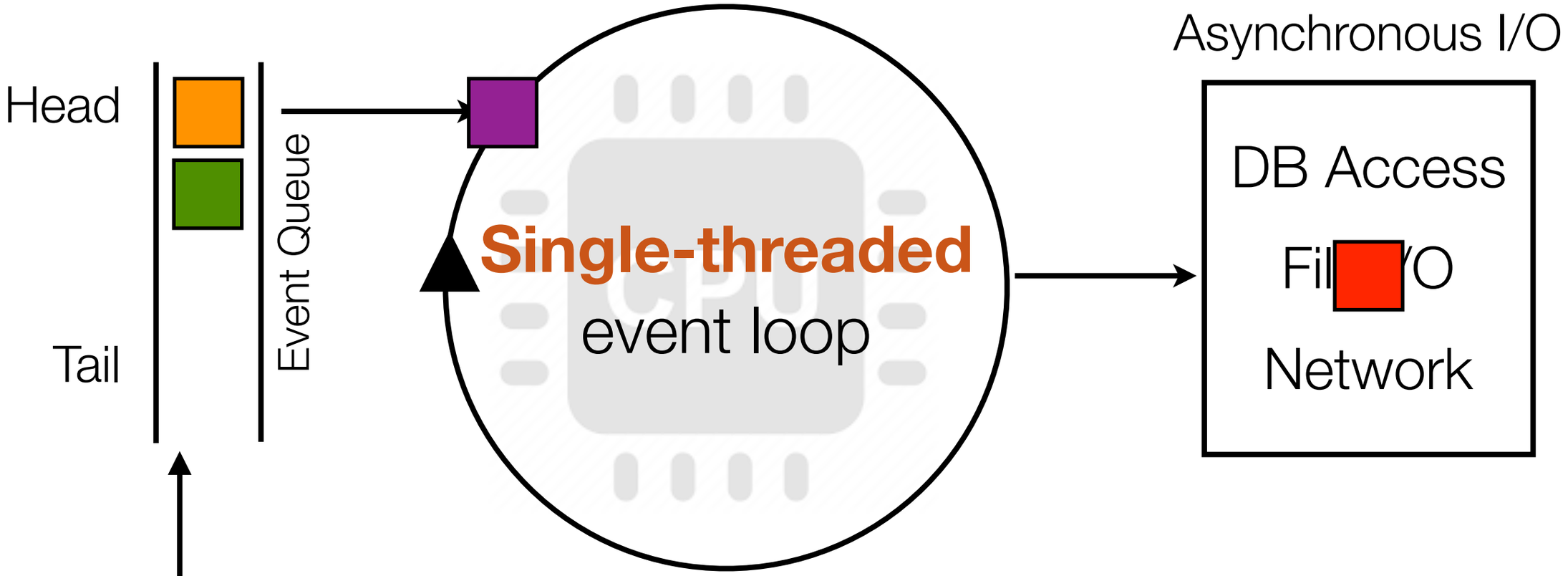

```
fs.readFile('input.txt',
  function (err, data) {
      if (err) return console.error(err);
      console.log(data.toString());
  }
);

console.log("Program continues…");
```

Event-driven Programming:
Emerging Approach



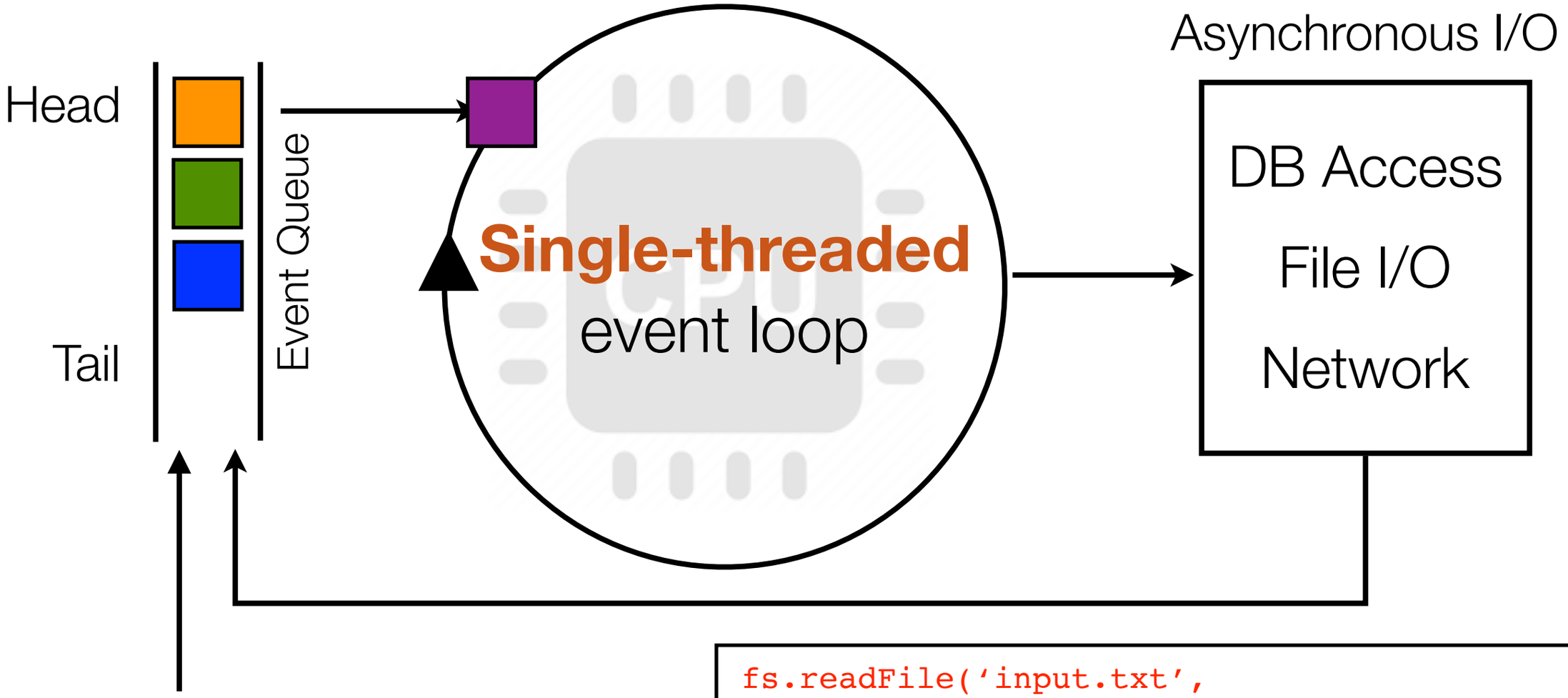

```
fs.readFile('input.txt',
  function (err, data) {
      if (err) return console.error(err);
      console.log(data.toString());
  }
);

console.log("Program continues…");
```

Event-driven Programming:
Emerging Approach




```
fs.readFile('input.txt',
  function (err, data) {
      if (err) return console.error(err);
      console.log(data.toString());
  }
);

console.log("Program continues…");
```

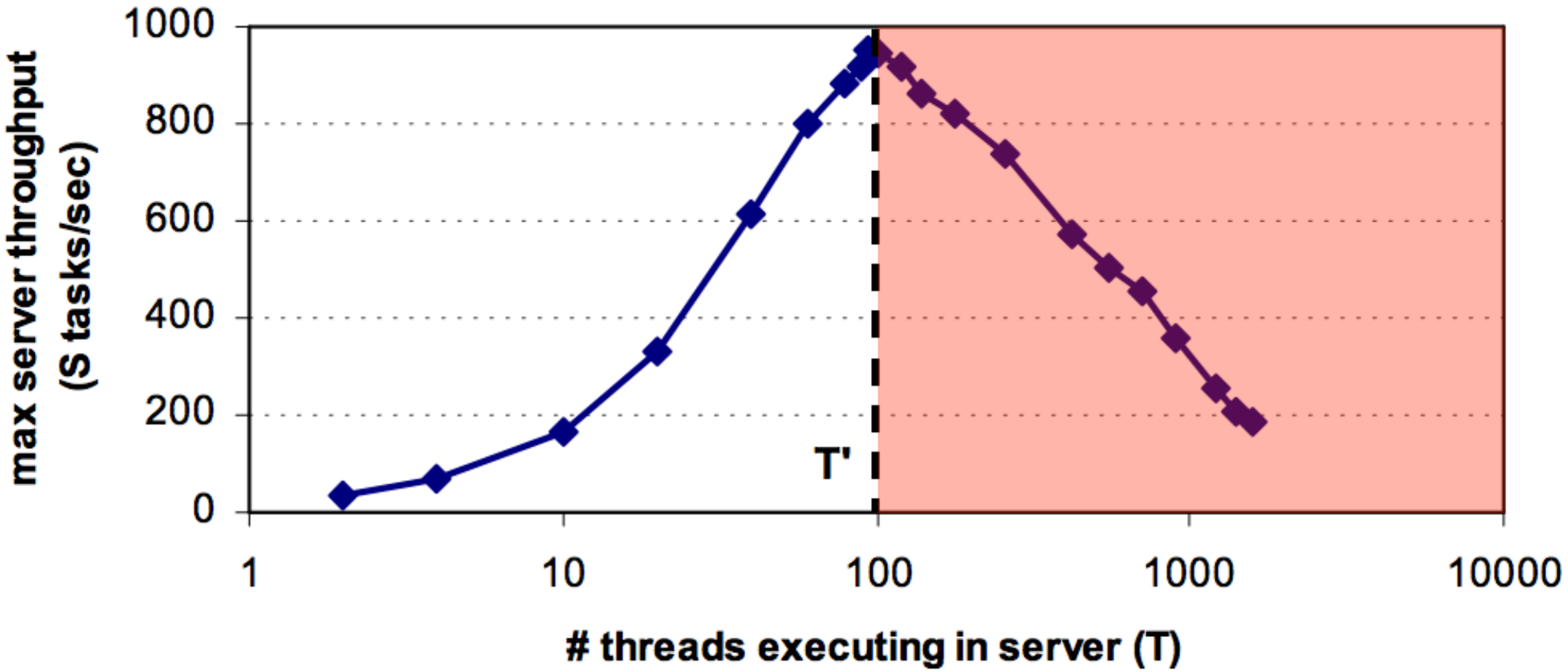
Event-driven Programming:
Emerging Approach



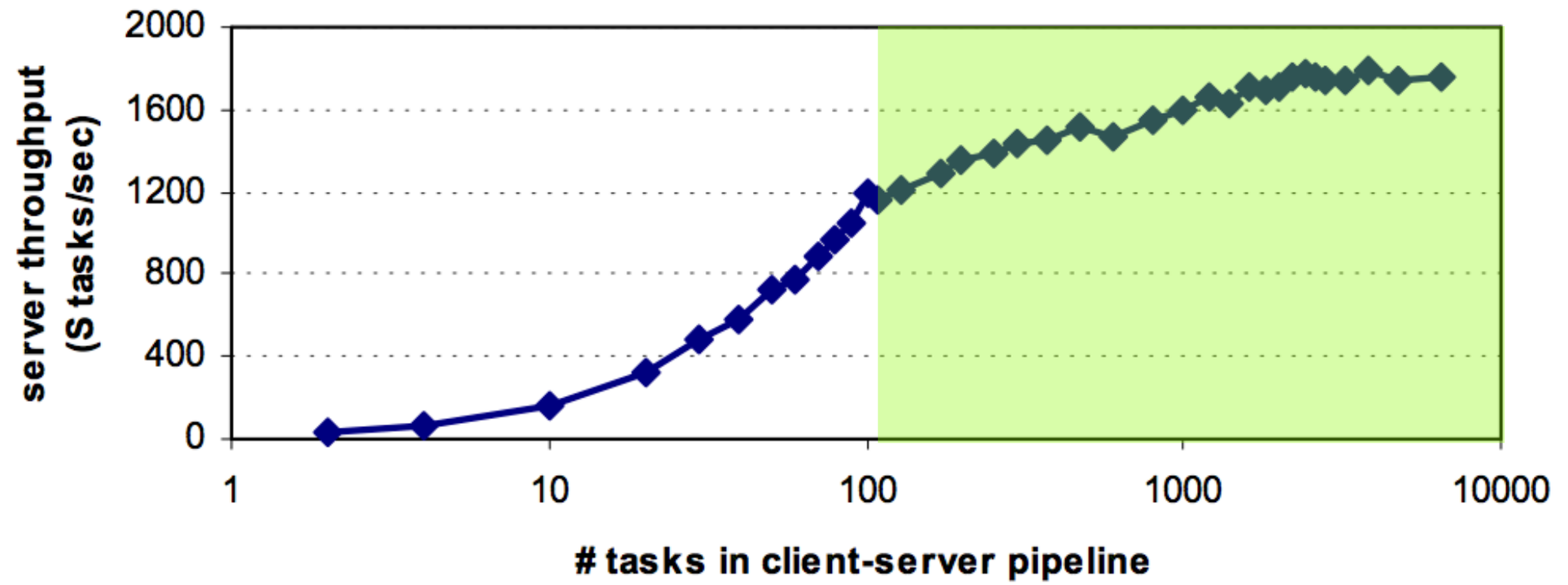

```
fs.readFile('input.txt',
  function (err, data) {
      if (err) return console.error(err);
      console.log(data.toString());
  }
);

console.log("Program continues…");
```

[Welsh et al. ’00]

Thread-based Programming

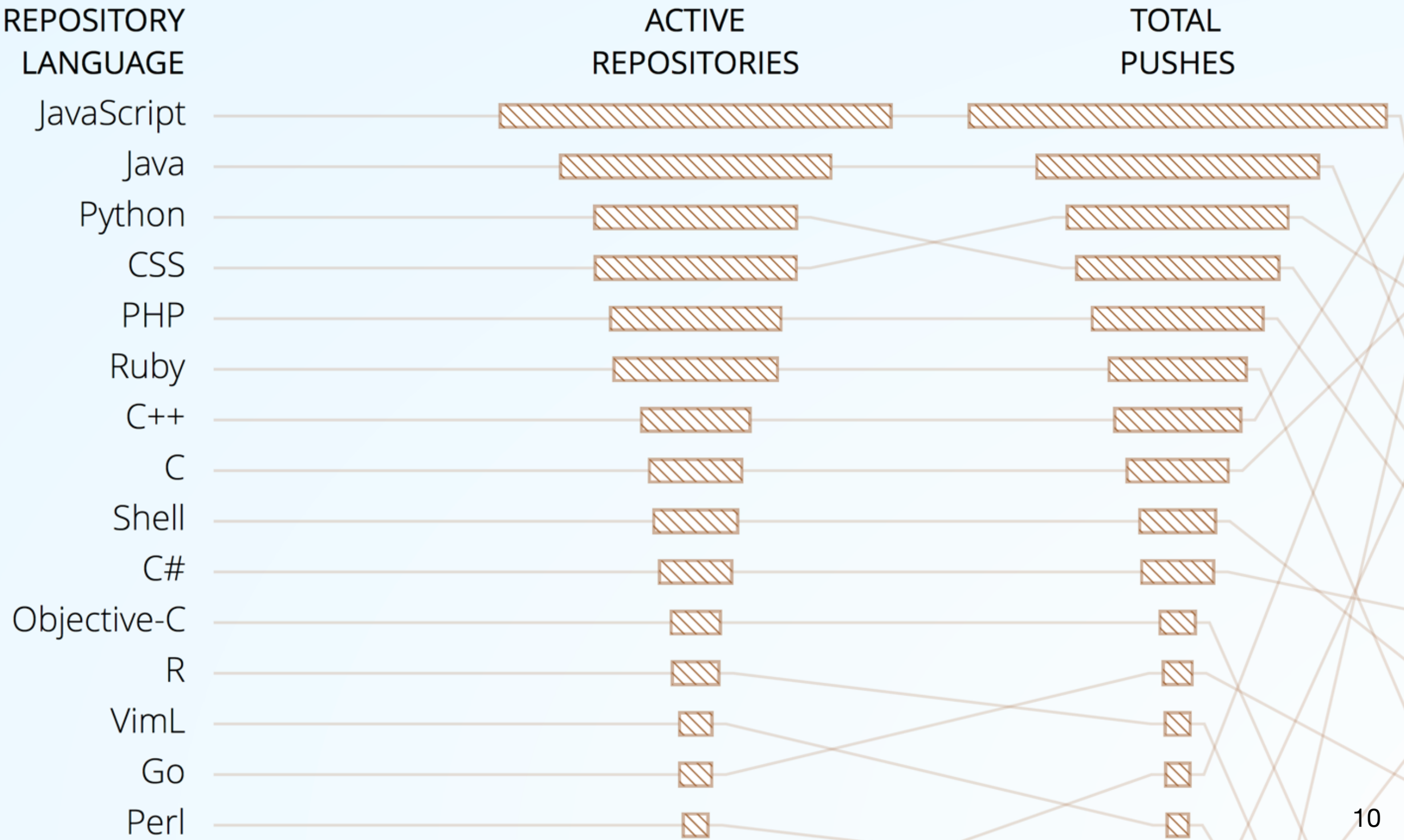


Event-driven Programming

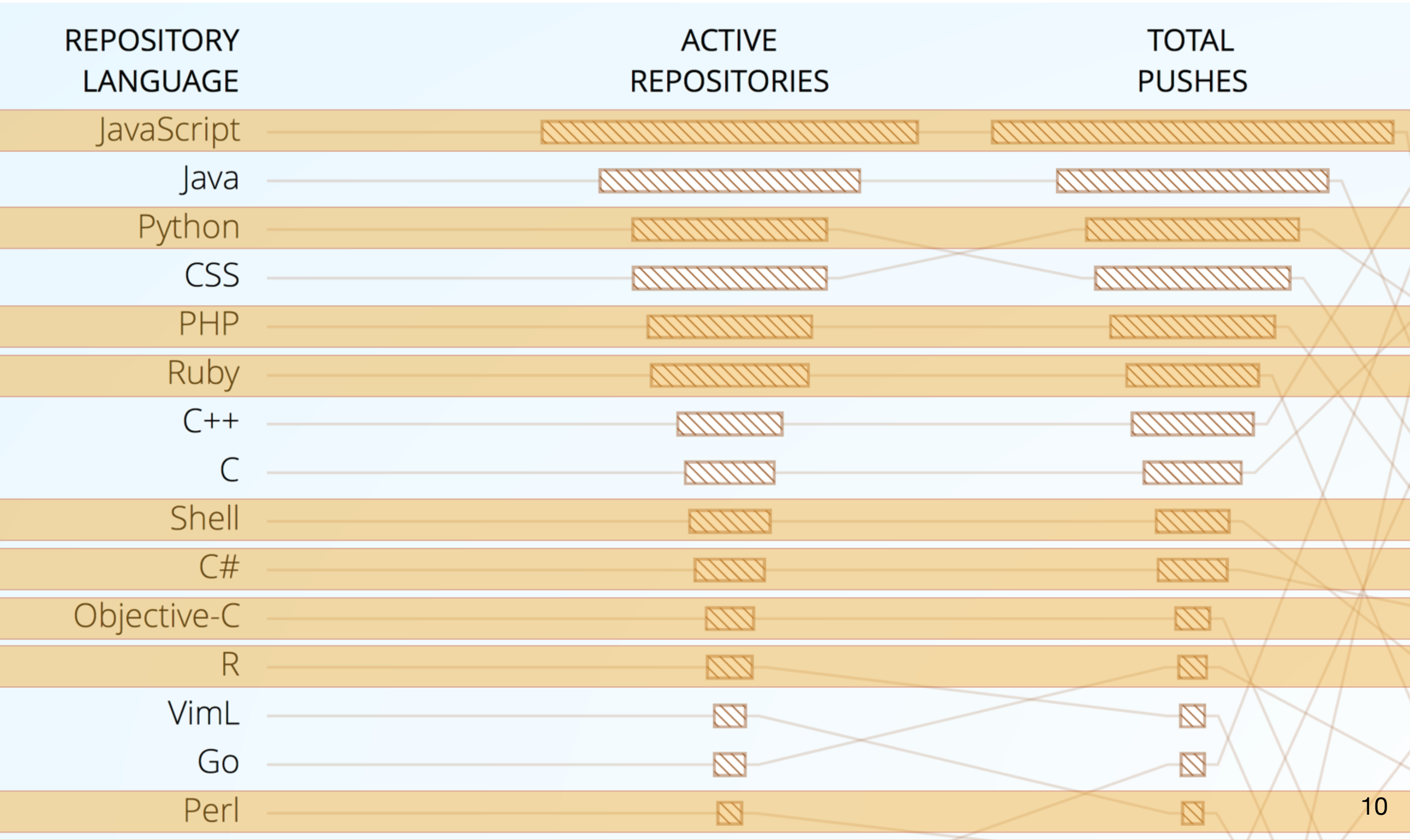


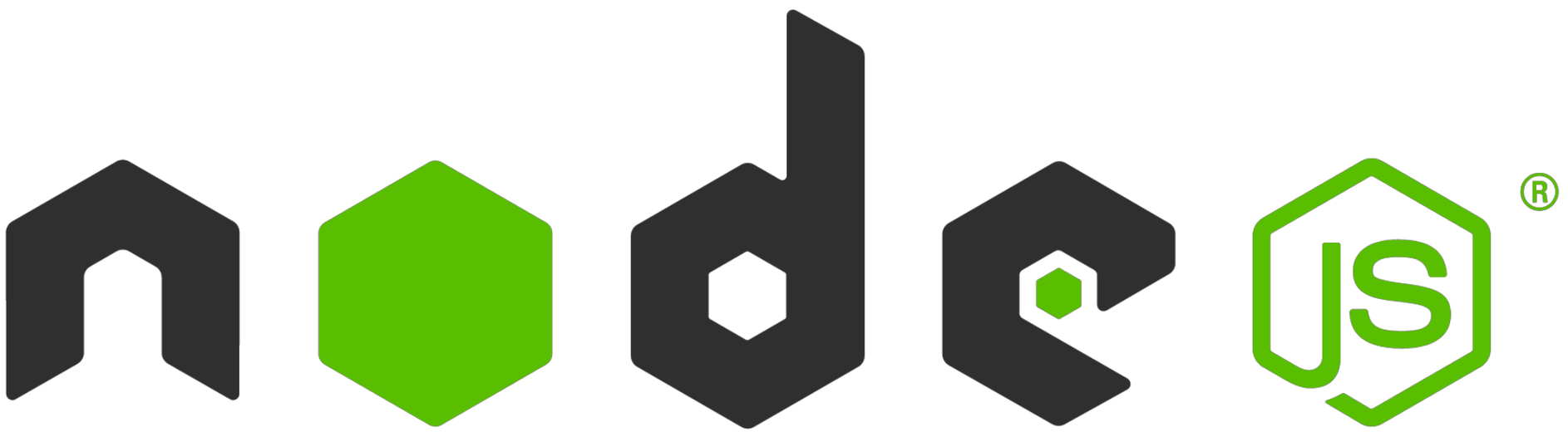
The Changing Programming Language Landscape

The Changing Programming Language Landscape

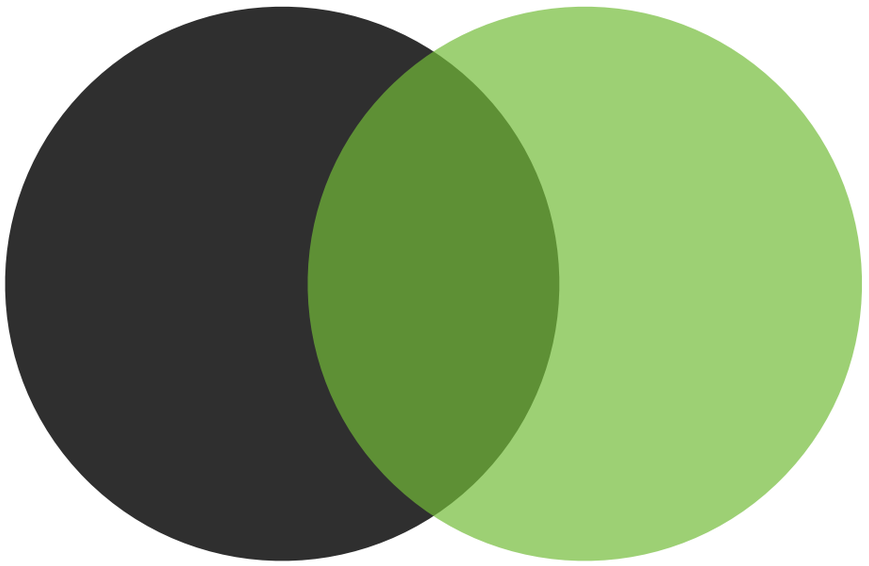


The Changing Programming Language Landscape





Event-driven Execution Model



Managed Language


PayPal

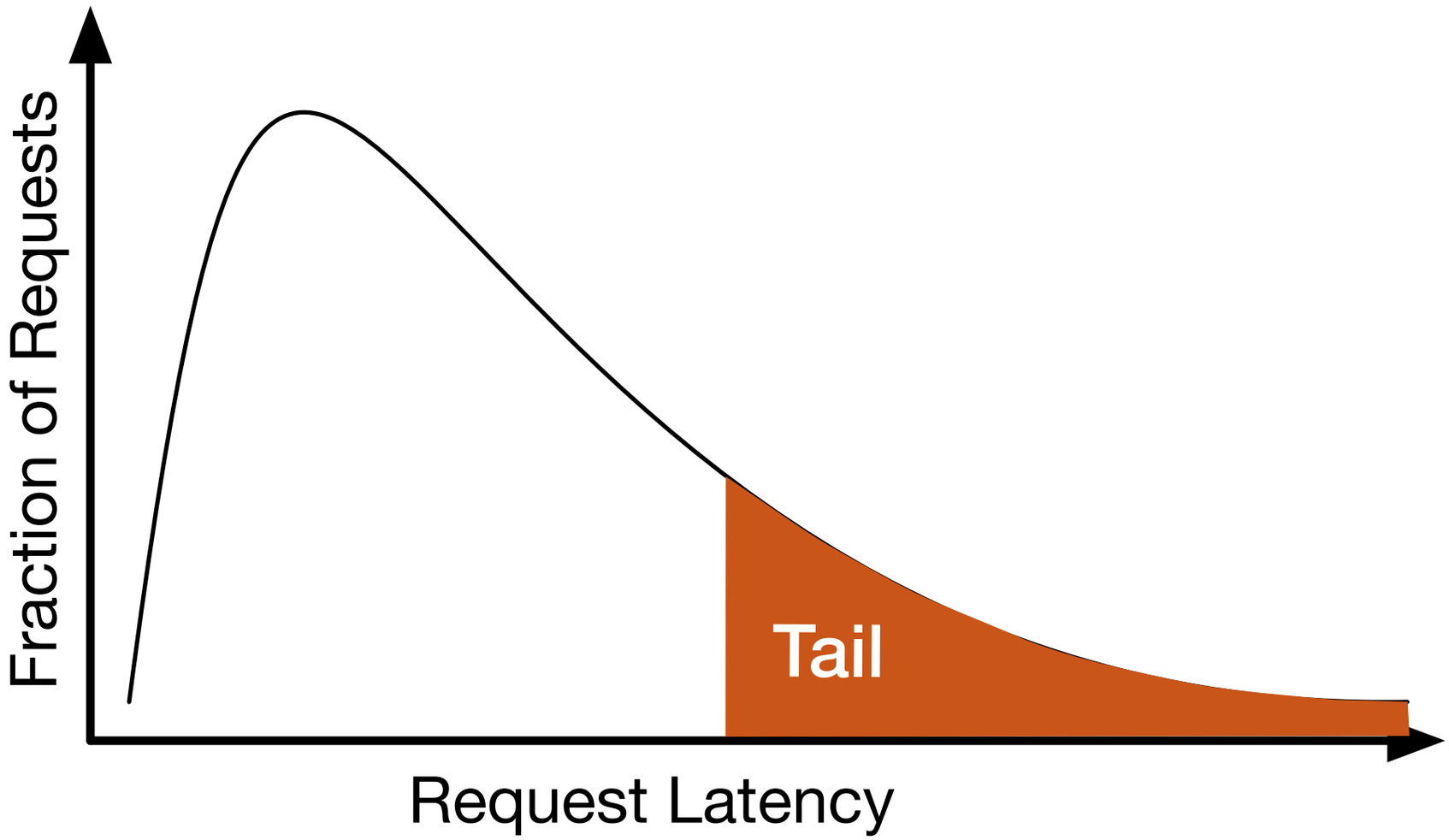

ebay


Linked in

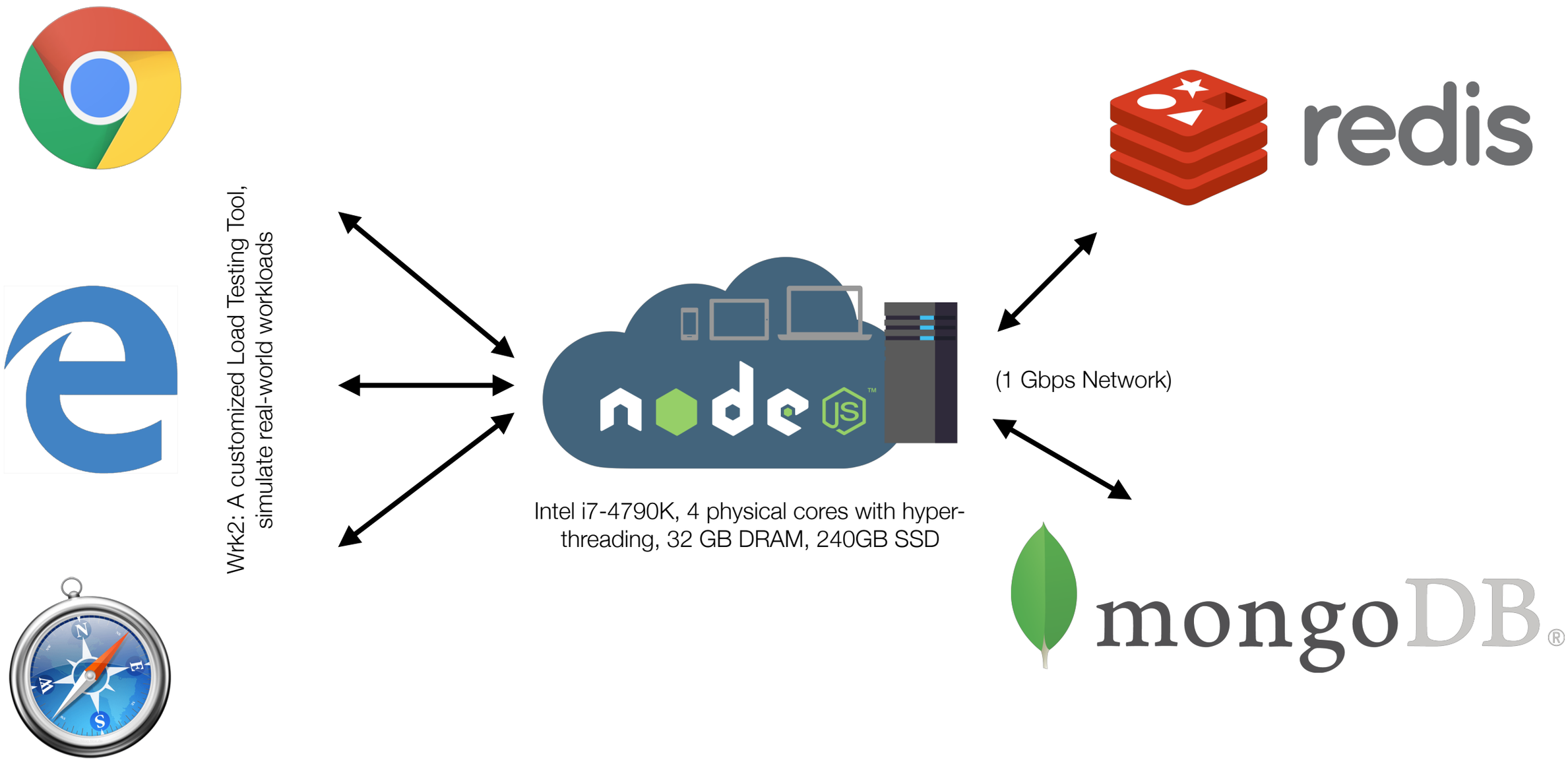

UBER


NETFLIX

Taming Tail Latencies in Event-Driven Web Services



Experimental Setup

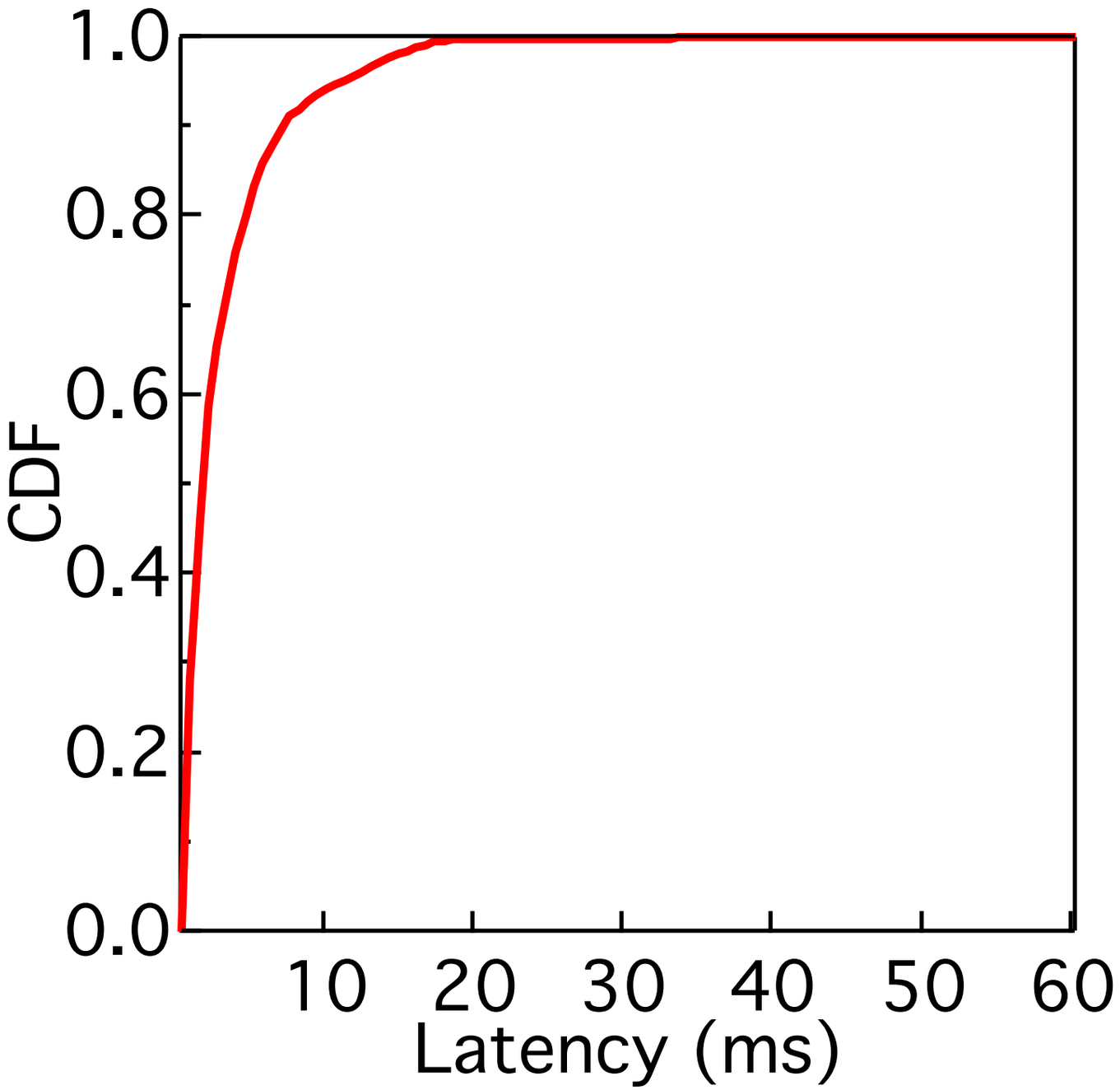


Applications

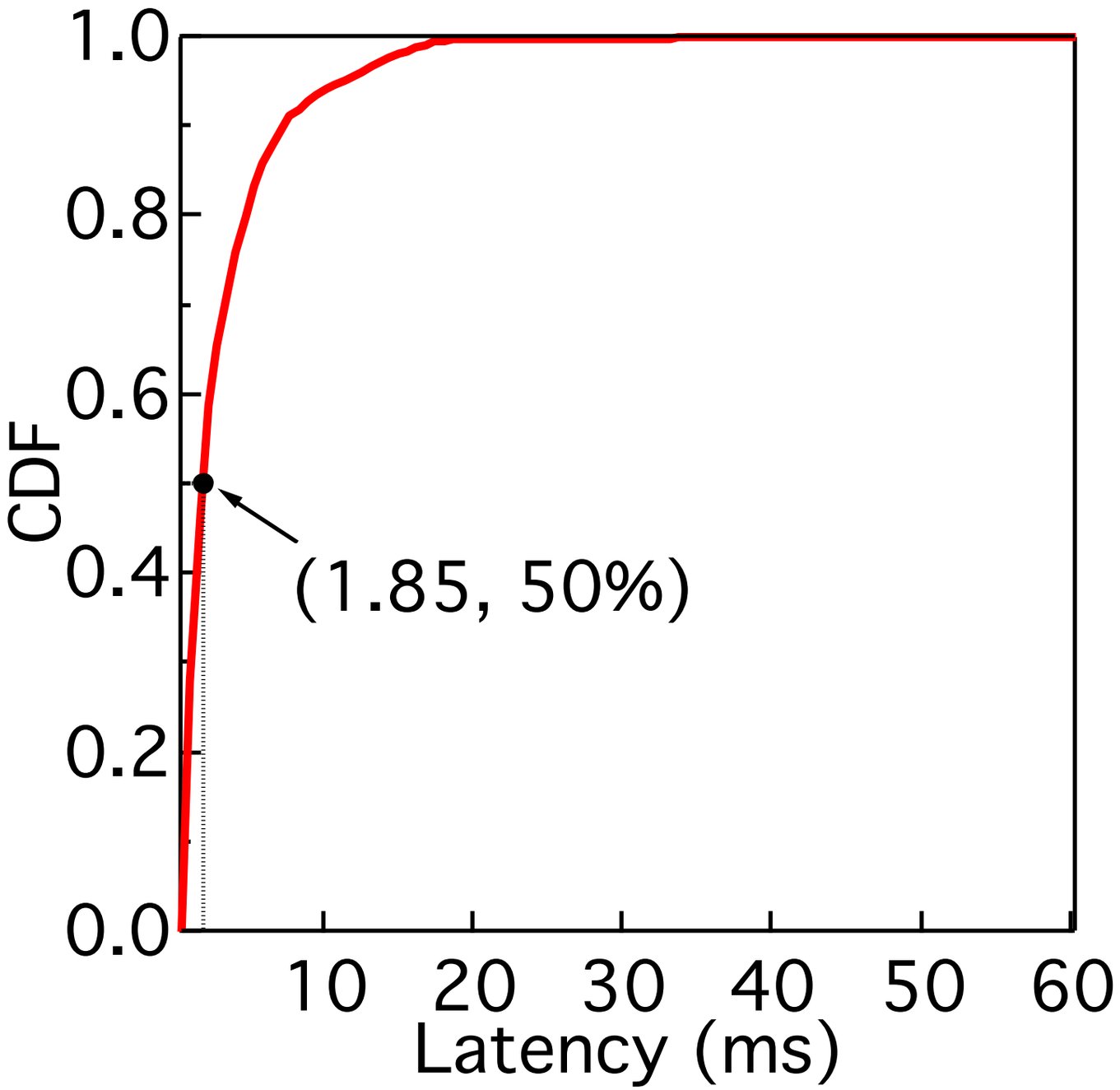
Benchmarks	I/O Type	#Requests	Description
Etherpad lite	N/A	20K	Real time word processor.
Todo	Redis	40K	Online Task Manager.
Lighter	Disk	40K	Blogging Engine.
Let’s Chat	MongoDB	10K	Web-based Chat Application.
Client Manager	MongoDB	40K	Online Address book.

Github Repo: https://github.com/nodebenchmark/benchmarks

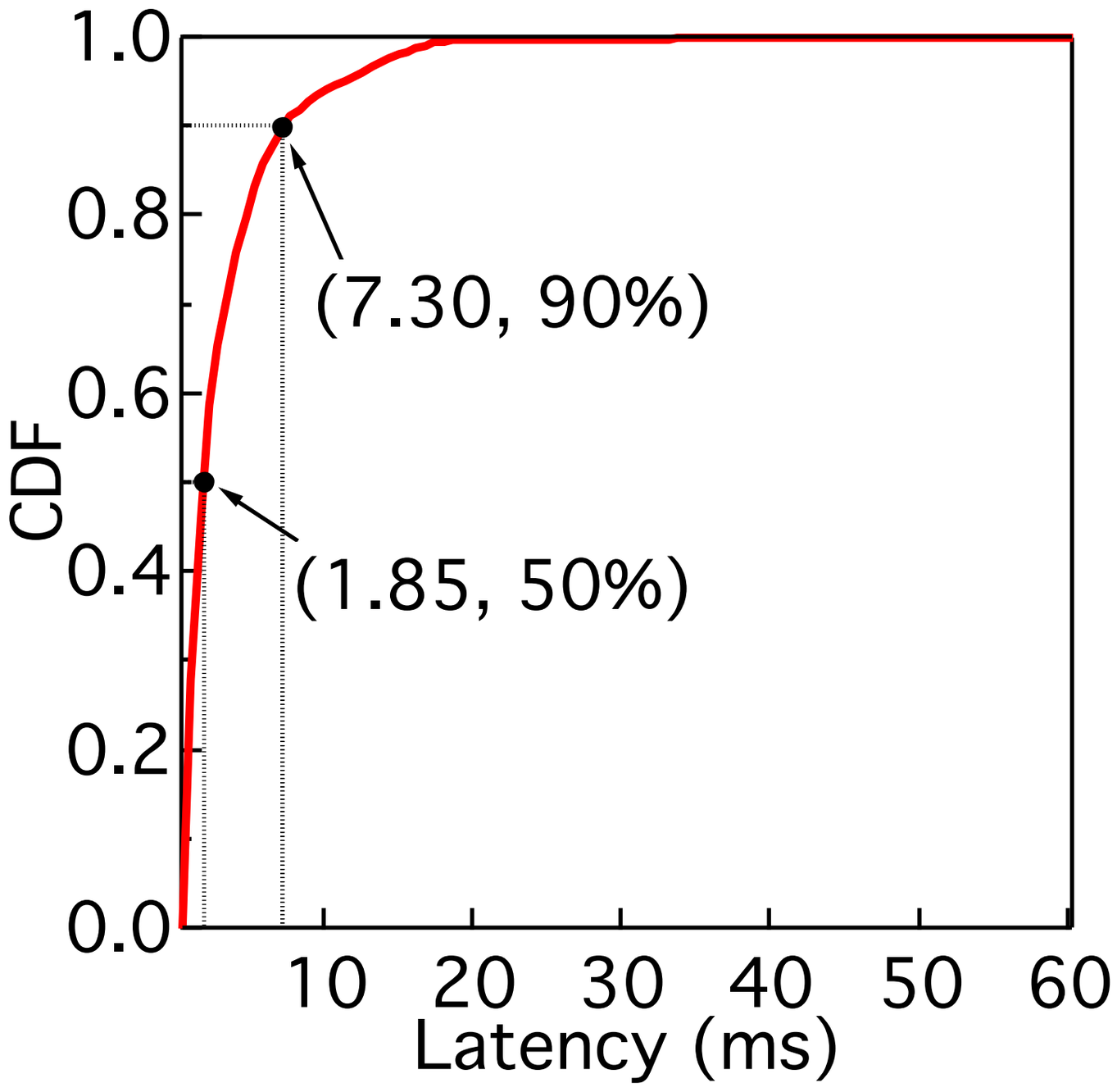
Etherpad: An Example



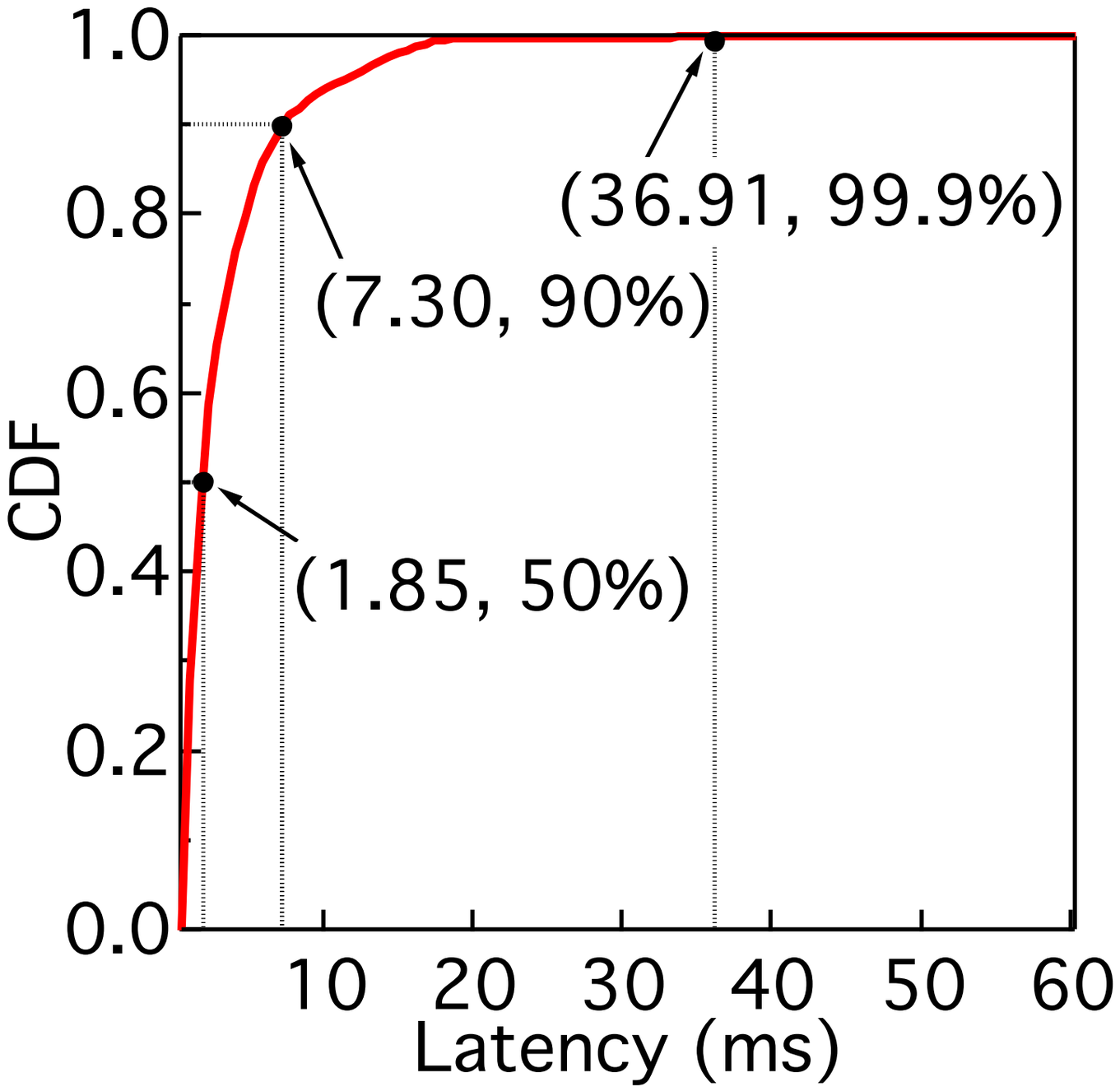
Etherpad: An Example



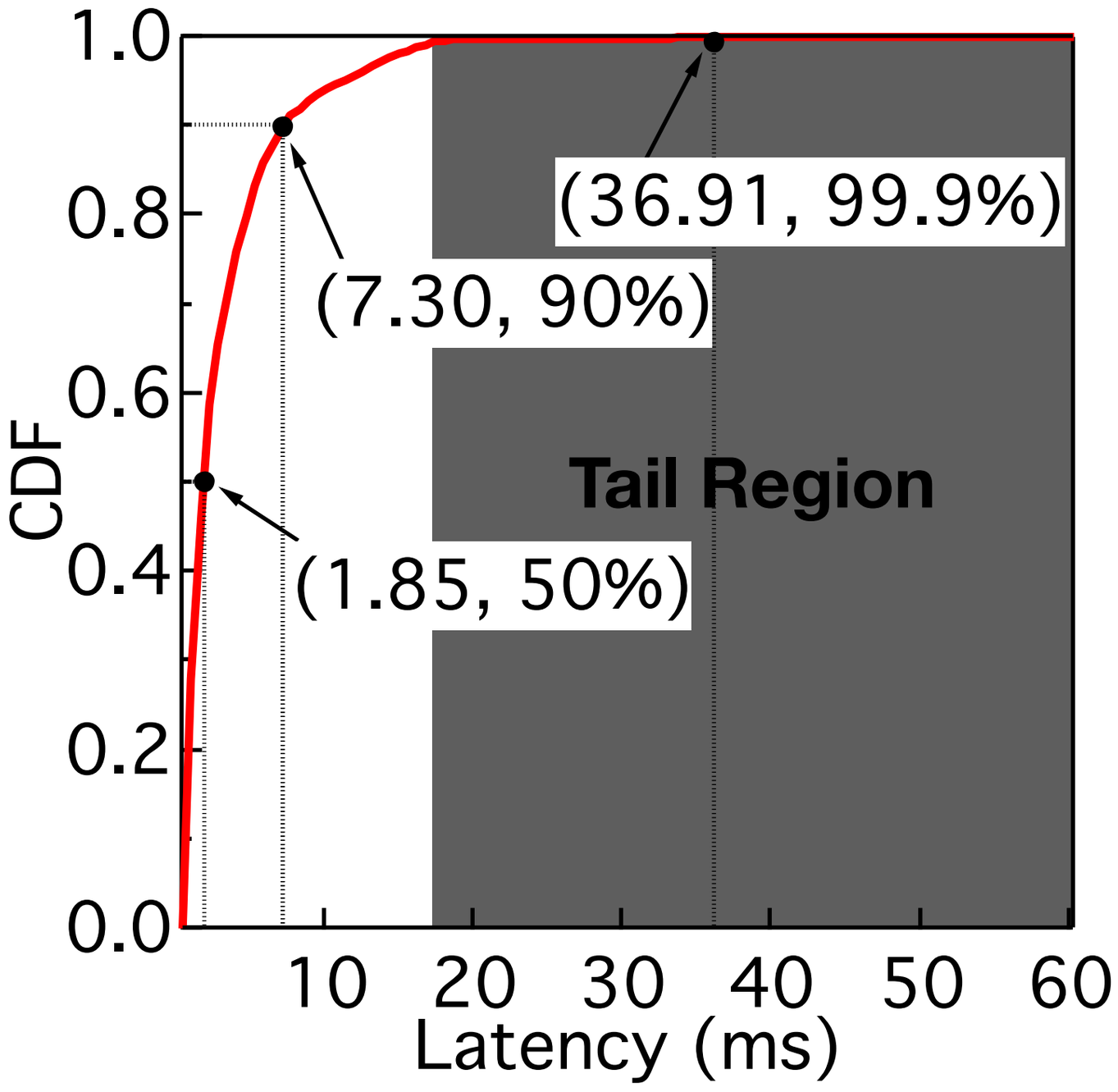
Etherpad: An Example

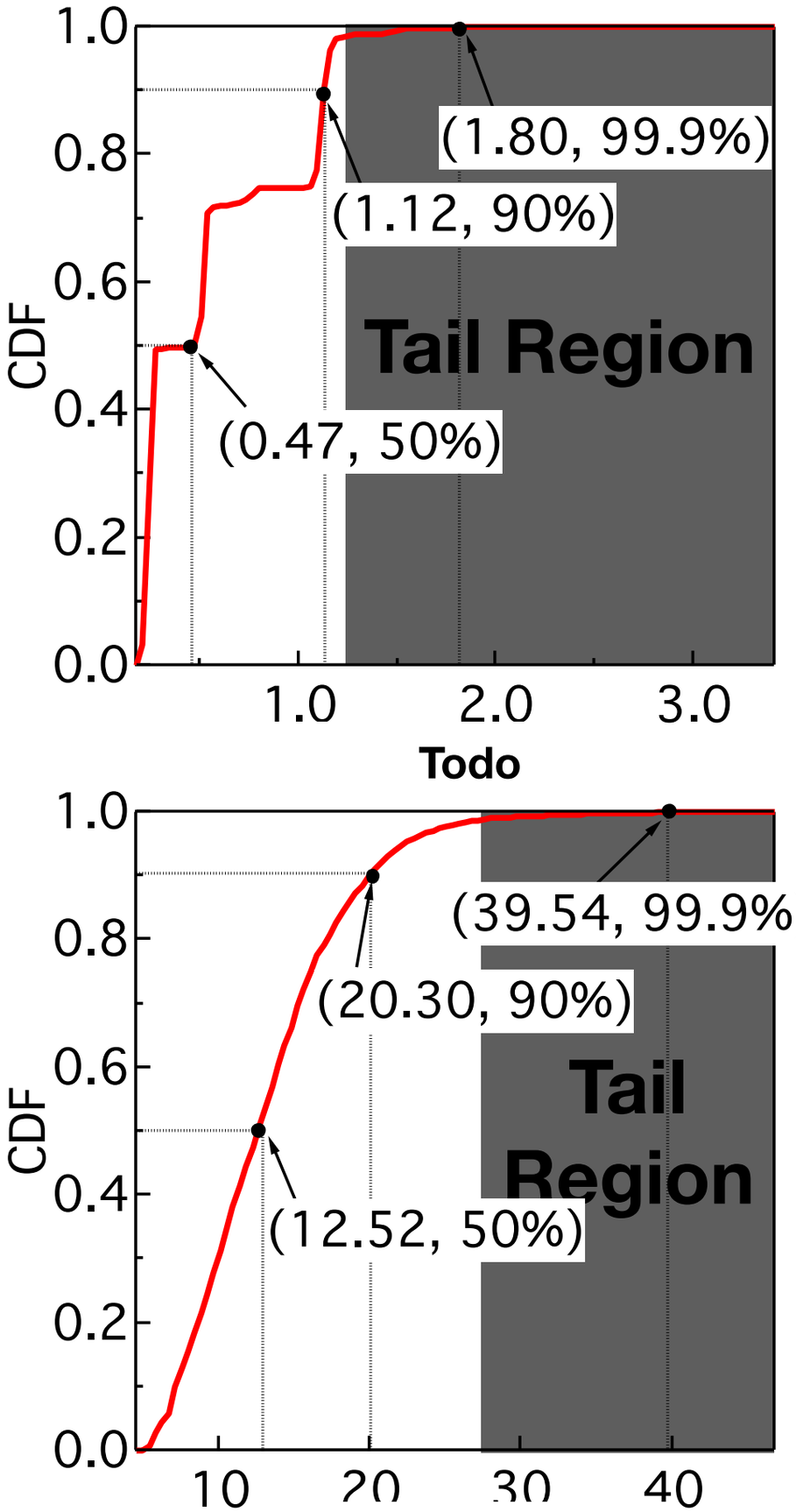


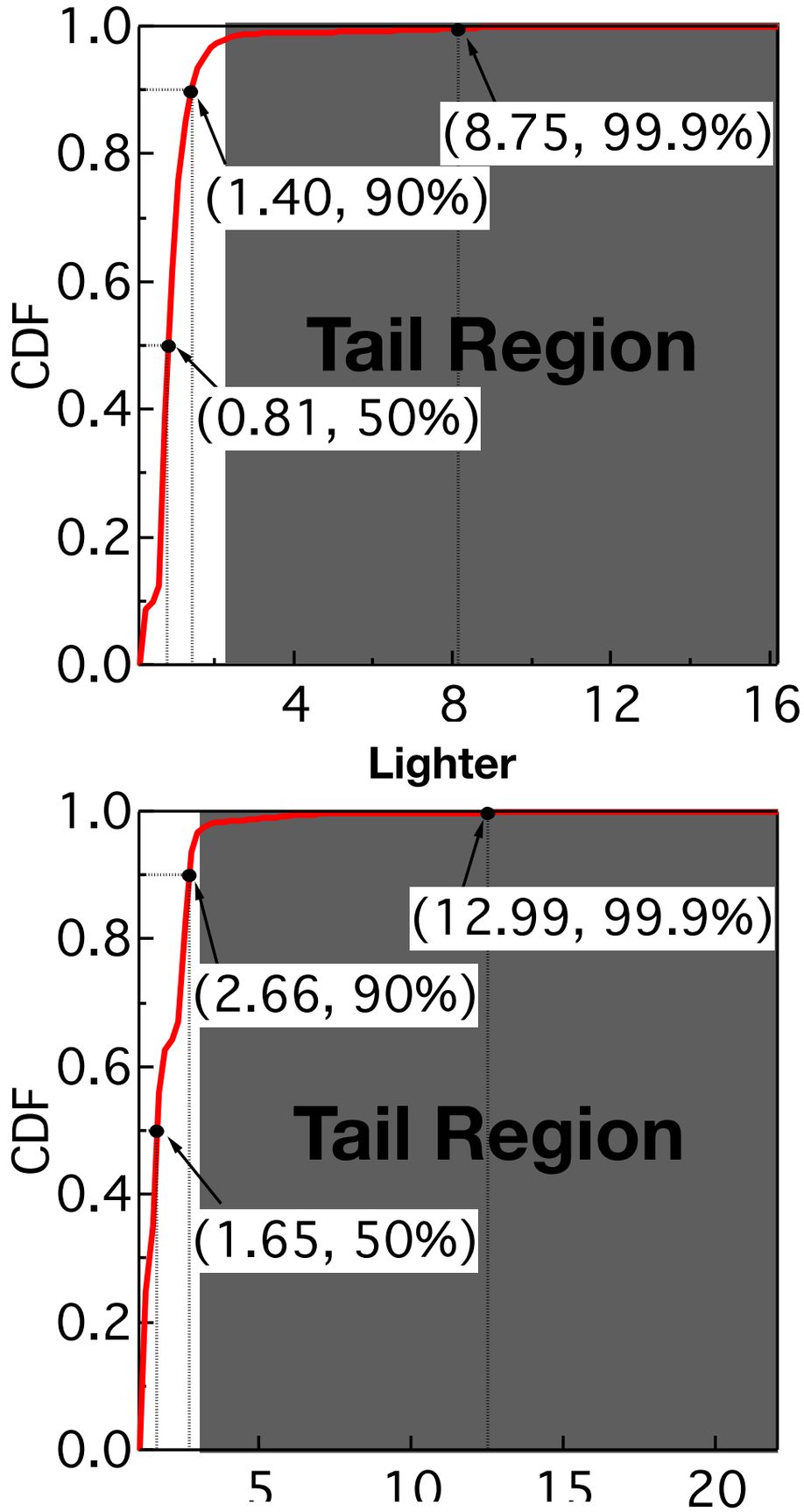
Etherpad: An Example



Etherpad: An Example

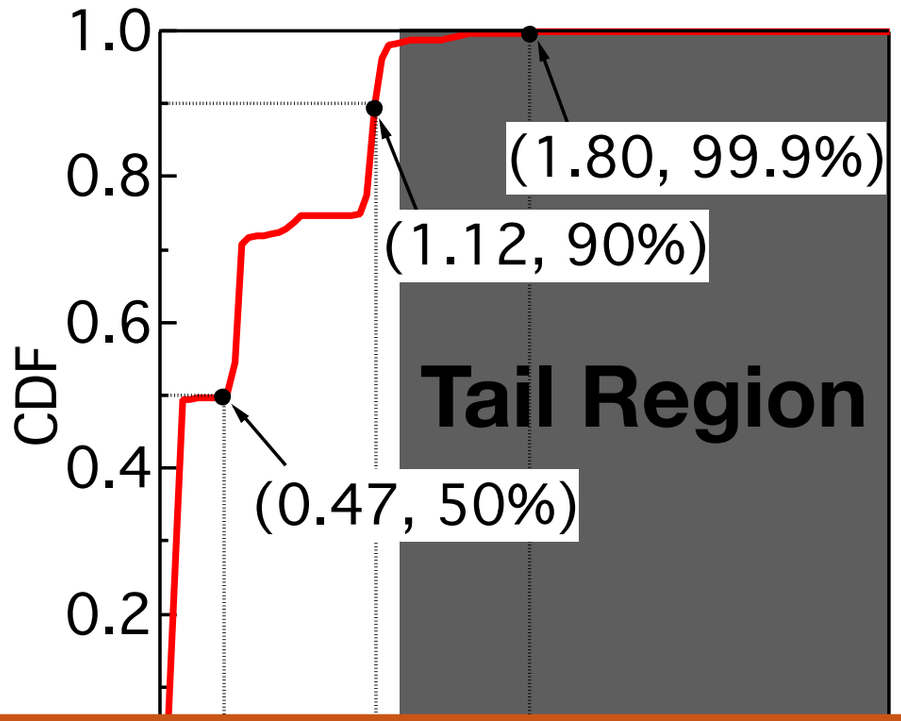


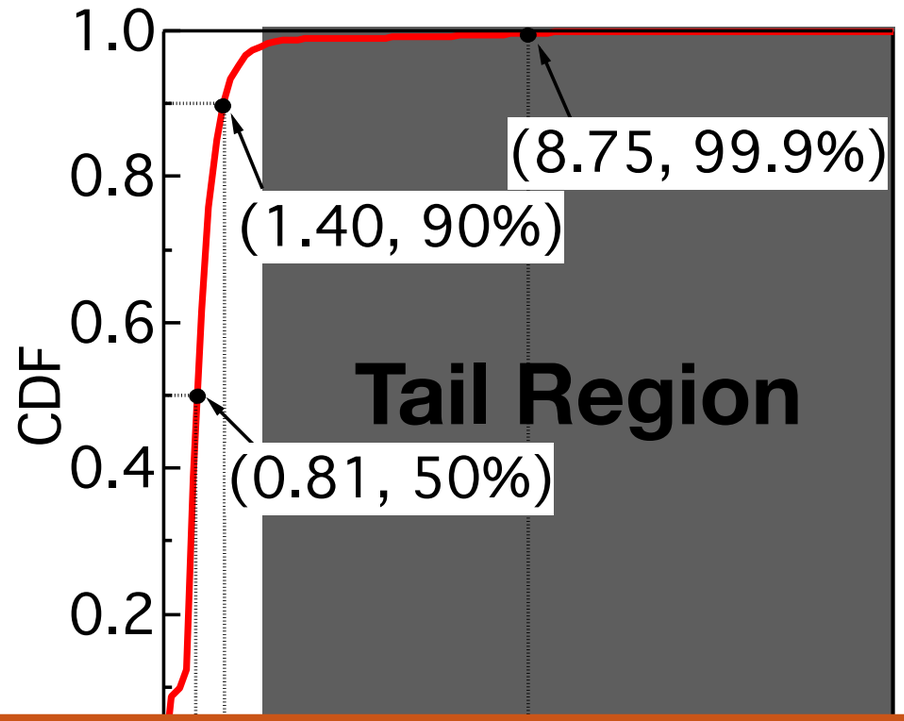

1.0
0.8
0.6
0.4
0.2
0.0
CDF
1.0
2.0
3.0
Todo
(1.80, 99.9%)
(1.12, 90%)
(0.47, 50%)
Tail Region


1.0
0.8
0.6
0.4
0.2
0.0
CDF
4
8
12
16
Lighter
(8.75, 99.9%)
(1.40, 90%)
(0.81, 50%)
Tail Region

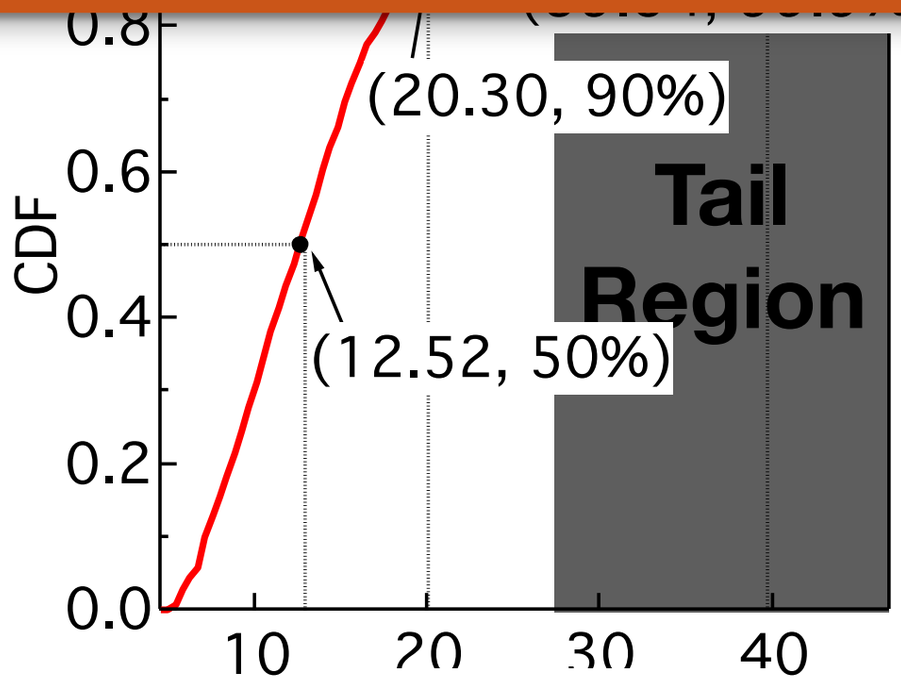
1.0
0.8
0.6
0.4
0.2
0.0
CDF
10
20
30
40
Let's Chat
(39.54, 99.9%)
(20.30, 90%)
(12.52, 50%)
Tail Region

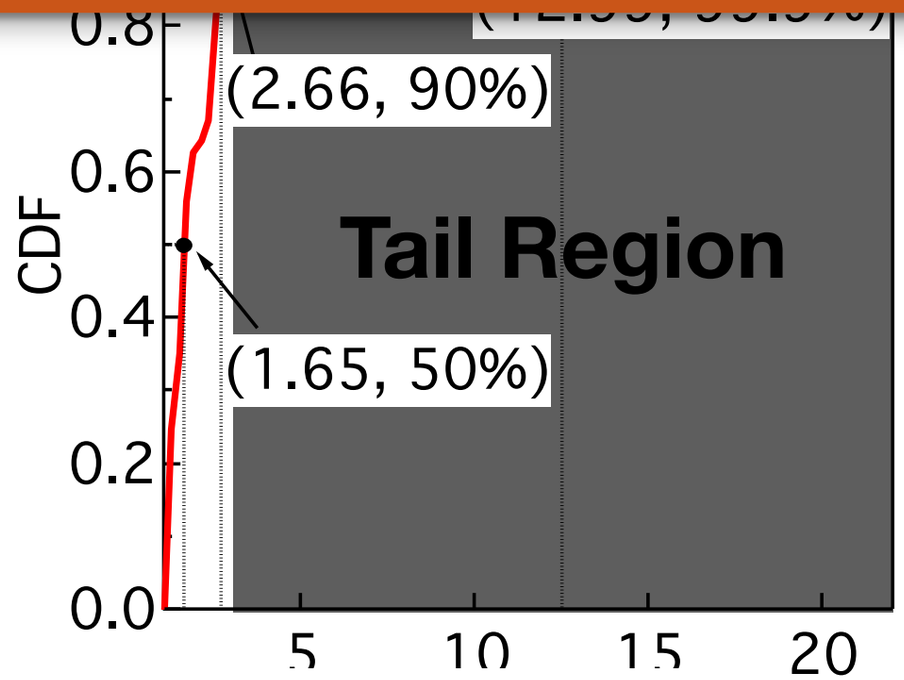
1.0
0.8
0.6
0.4
0.2
0.0
CDF
5
10
15
20
Client Manager
(12.99, 99.9%)
(2.66, 90%)
(1.65, 50%)
Tail Region





Tail latency (99.9%) is 9.1x longer than median request latency





System Overview

Step 1 **Step 2** **Step 3**

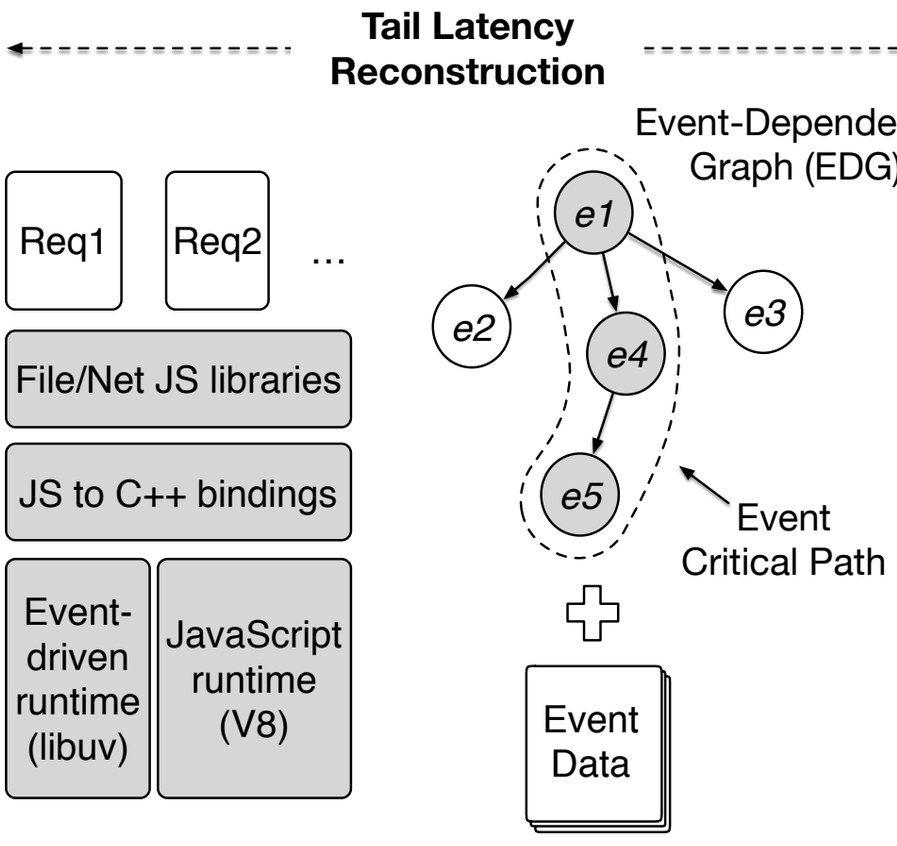
System Overview

Step 1 **Step 2** **Step 3**

Tools to Root-cause Tail in Node.js

System Overview

Step 1 **Step 2** **Step 3**

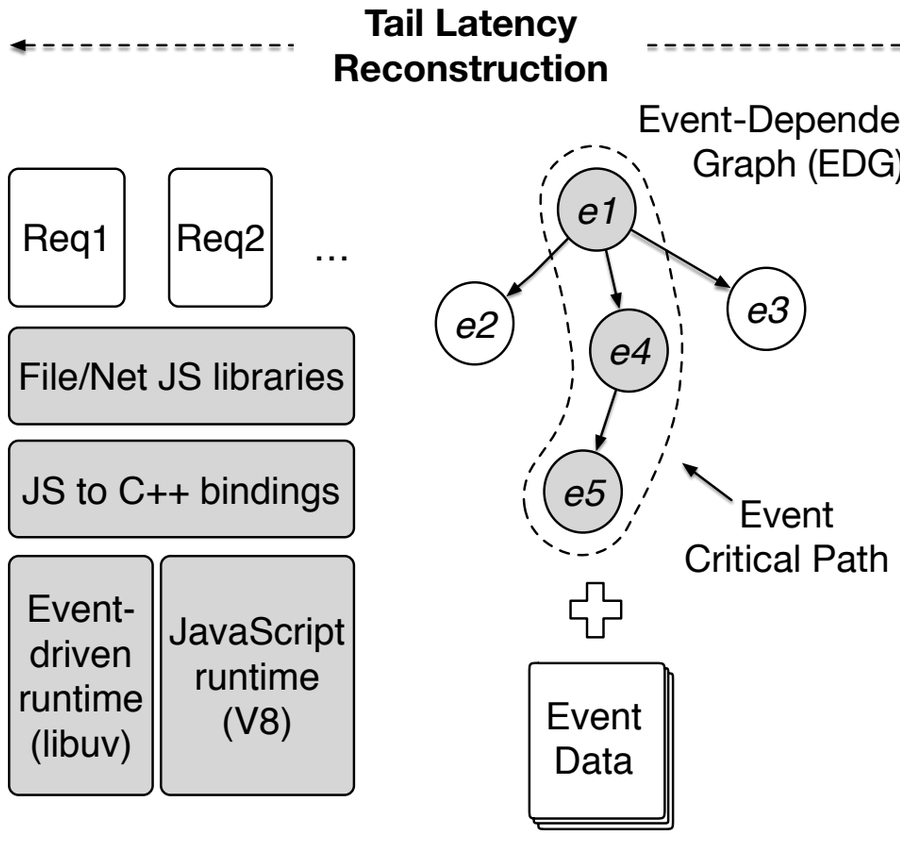


System Overview

Step 1

Step 2

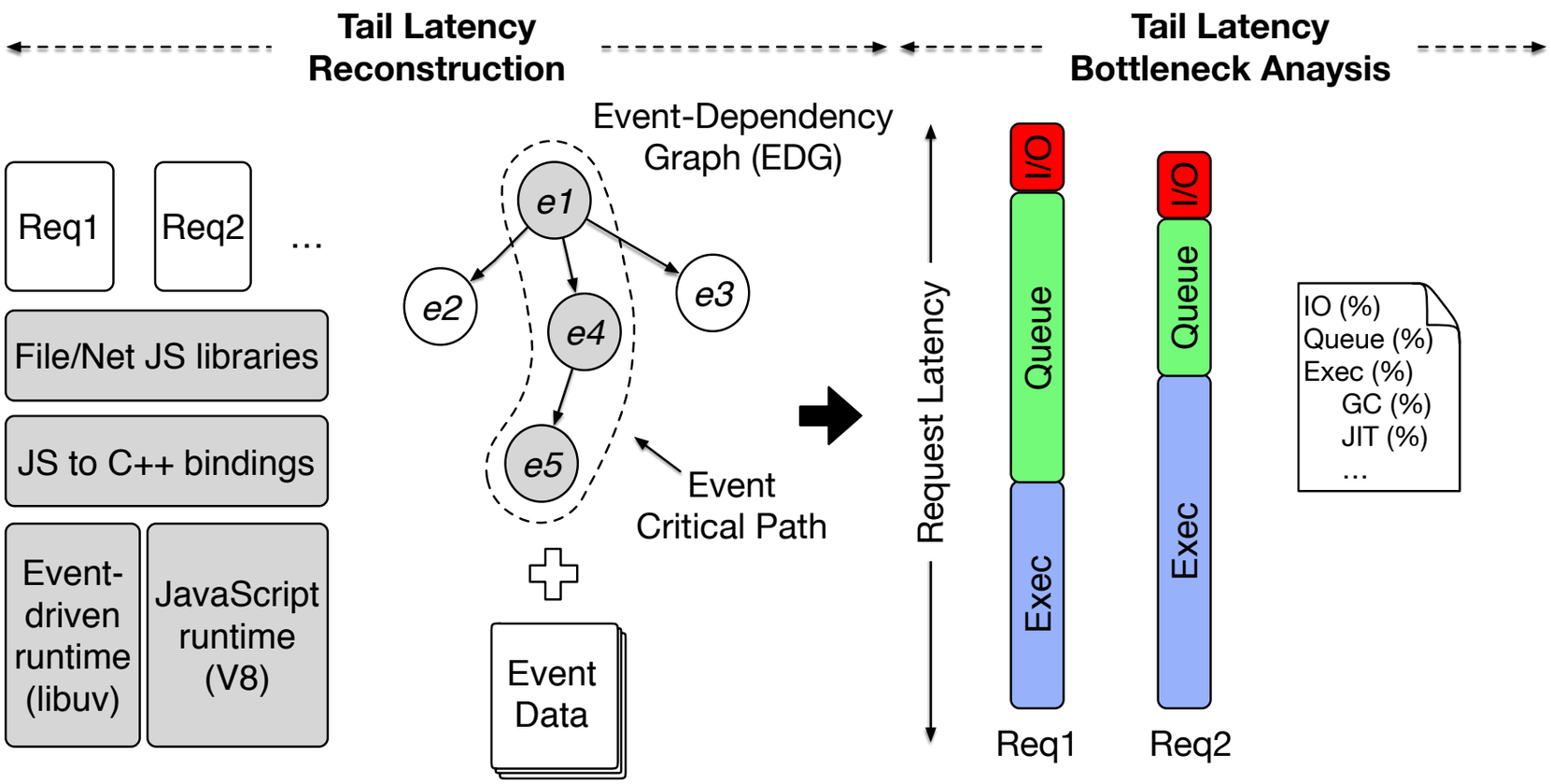
Step 3



Root-causing Tail in Node.js

System Overview

Step 1 **Step 2** **Step 3**

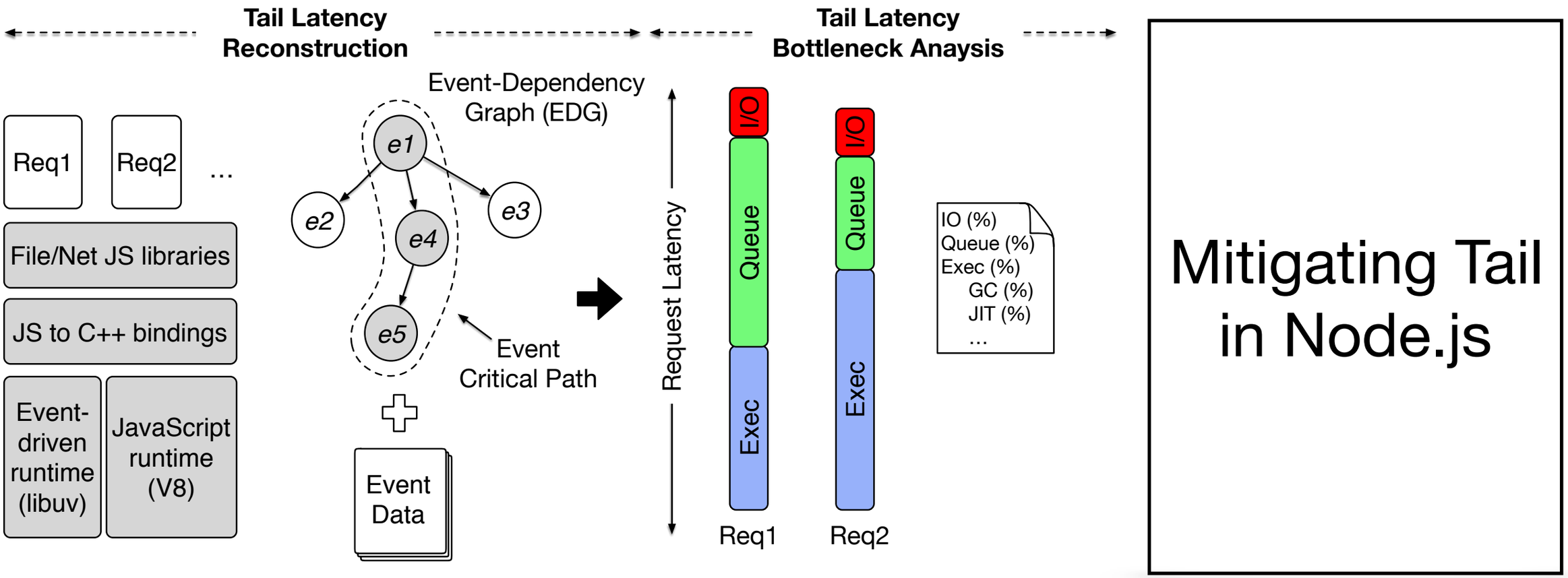


System Overview

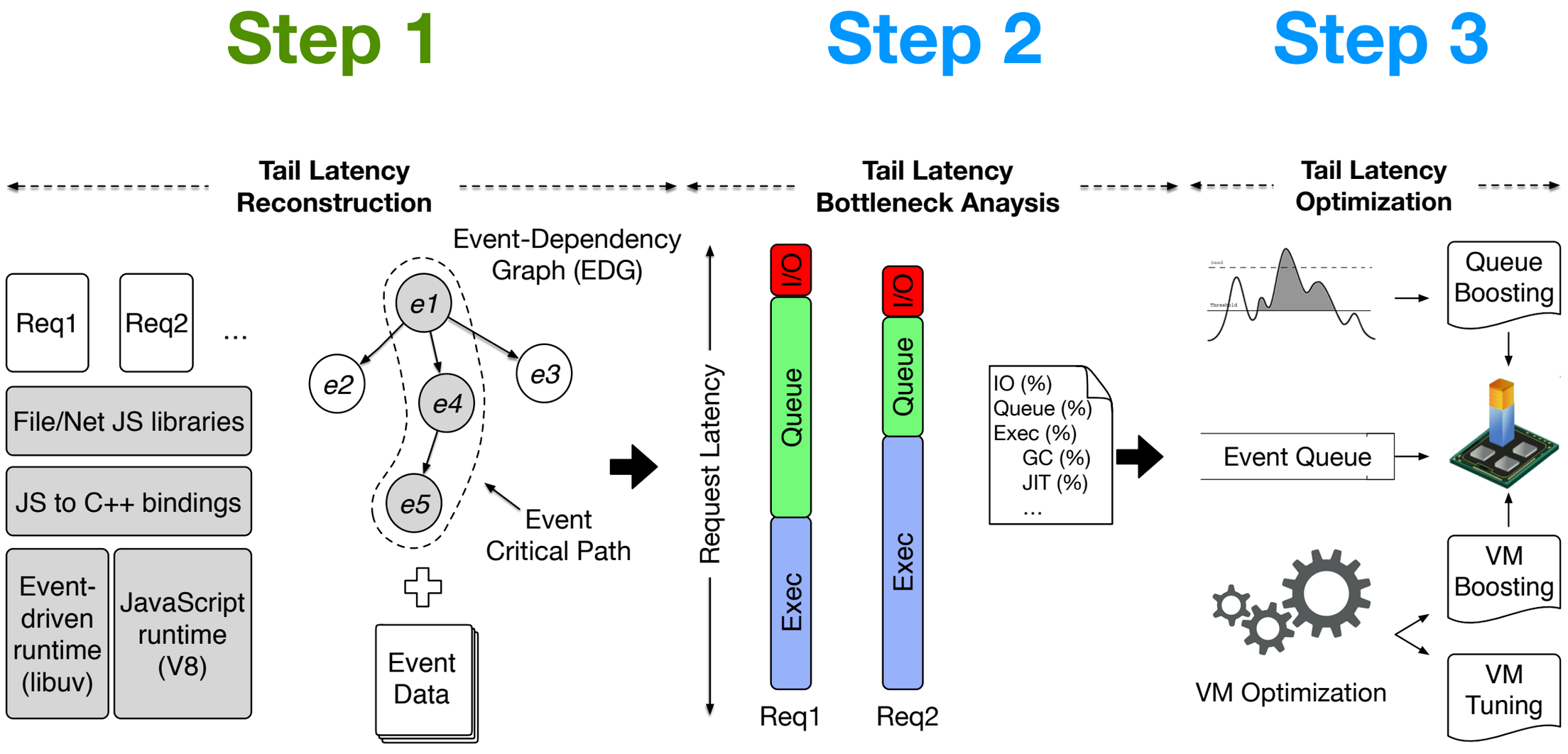
Step 1

Step 2

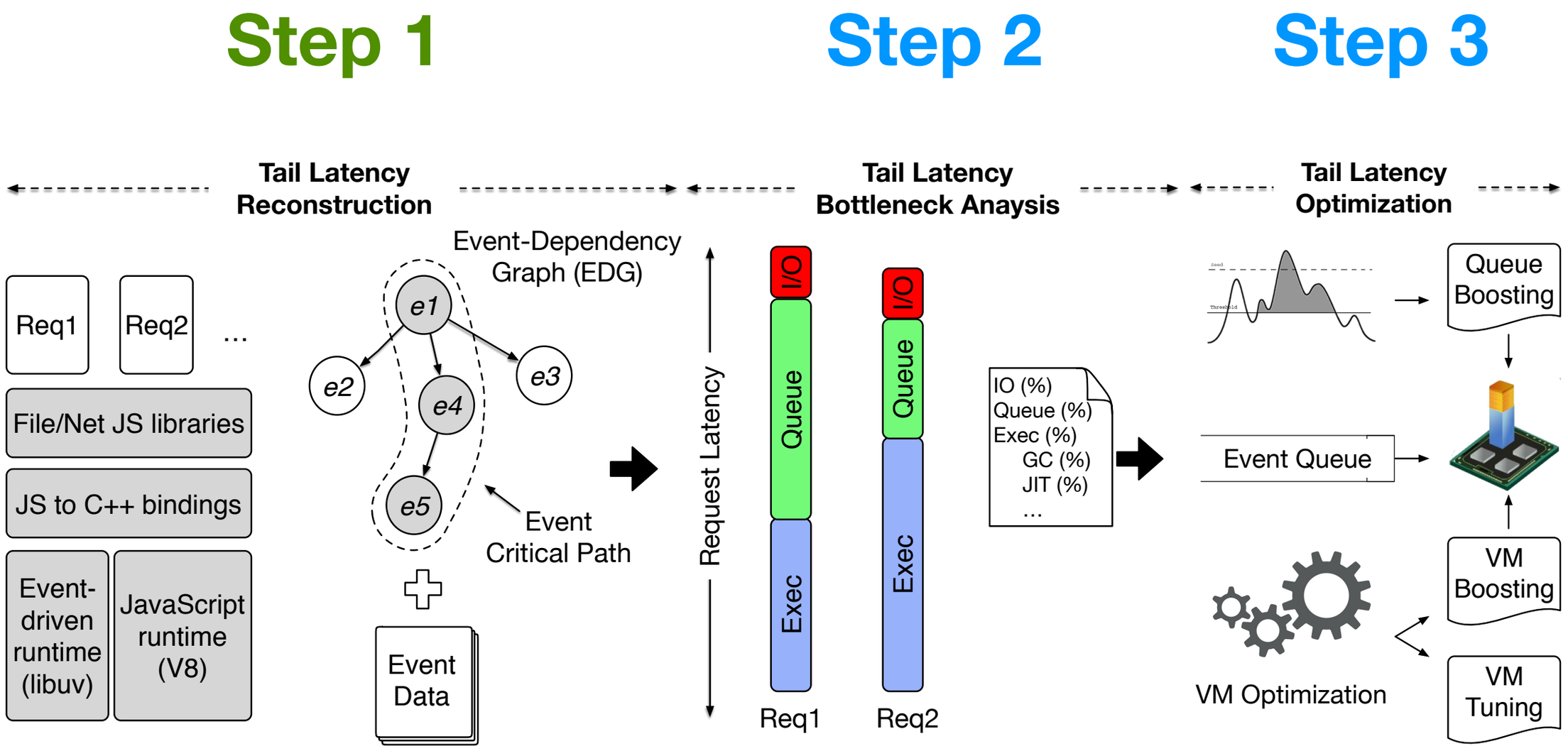
Step 3



System Overview

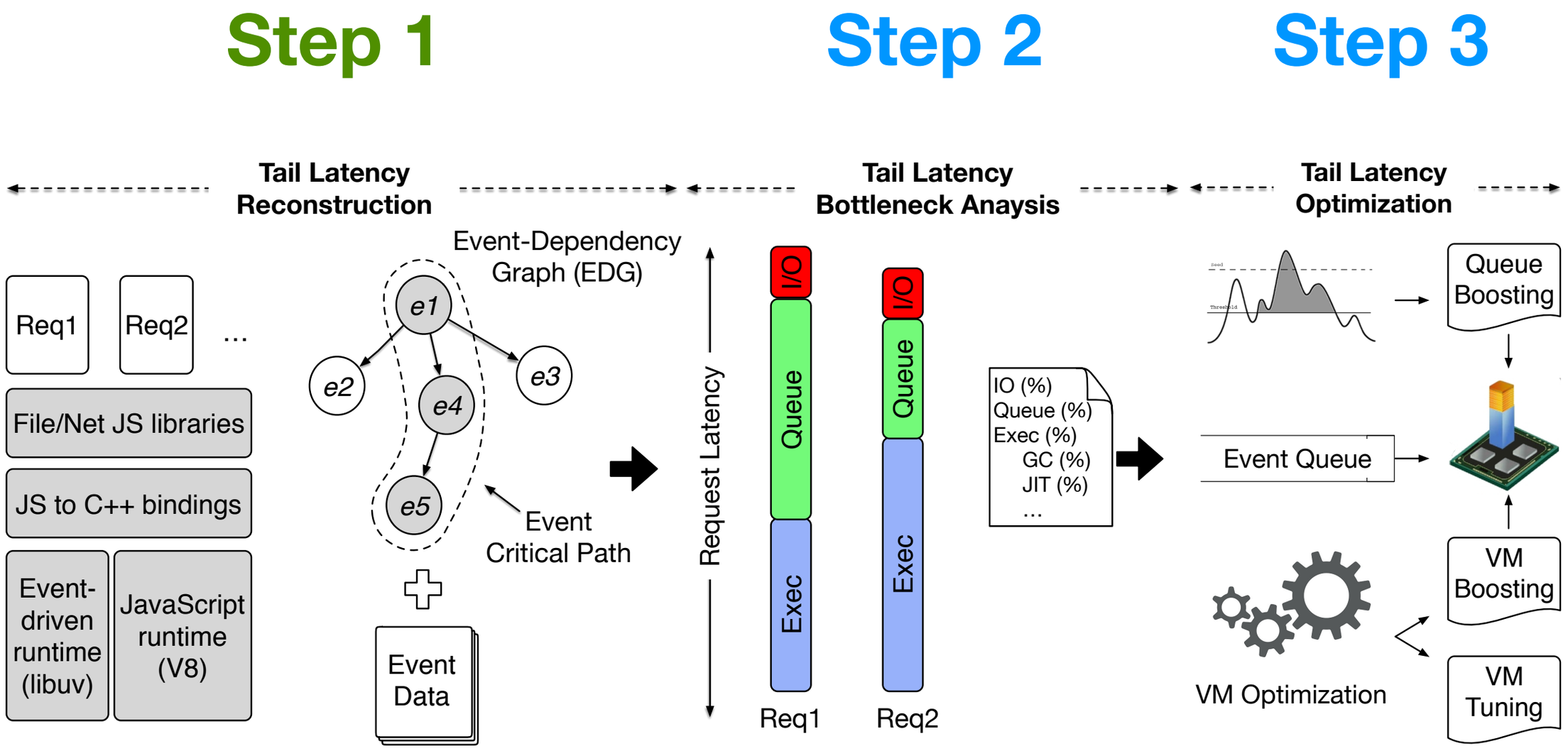


System Overview



Static Instrumentation

System Overview



Static Instrumentation

Dynamic Analysis & Optimization

Step 1: Latency Reconstruction

```
var count = N;
fs.readdir("/data", function dir(err, files) {
});
```

Step 1: Latency Reconstruction

```
var count = N;
fs.readdir("/data", function dir(err, files) {
  files.foreach(function file(f, index) {
  });
});
```

Step 1: Latency Reconstruction

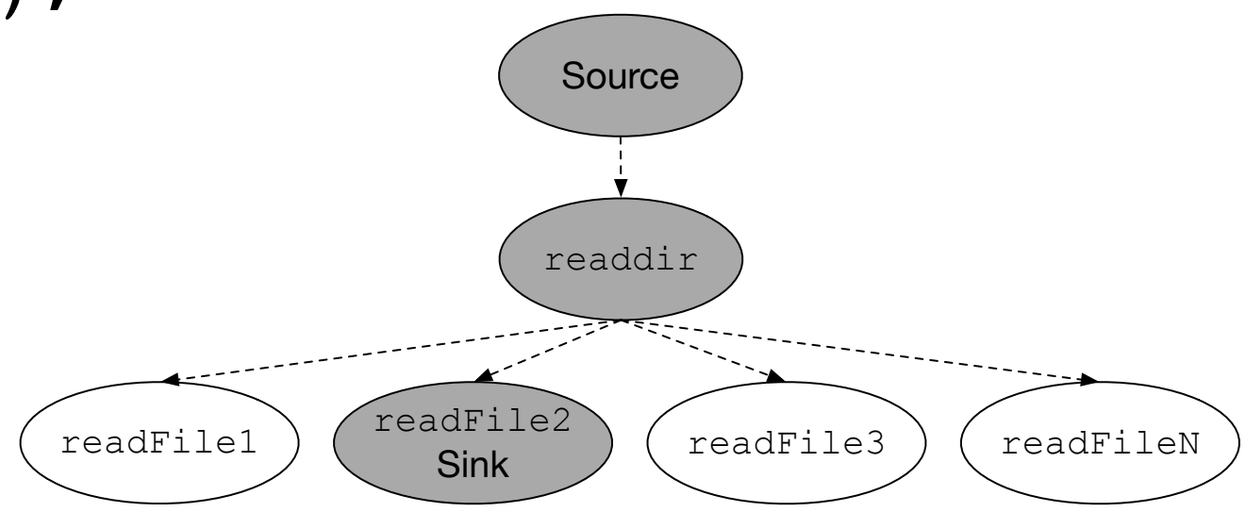
```
var count = N;
fs.readdir("/data", function dir(err, files) {
  files.foreach(function file(f, index) {
    var fname = …;
    fs.readFile(fname, function read(err, data) {
    });
  });
});
```

Step 1: Latency Reconstruction

```
var count = N;
fs.readdir("/data", function dir(err, files) {
  files.foreach(function file(f, index) {
    var fname = …;
    fs.readFile(fname, function read(err, data) {
      count -= 1;
      if (count == 0)
        sendResponse();
    });
  });
});
```

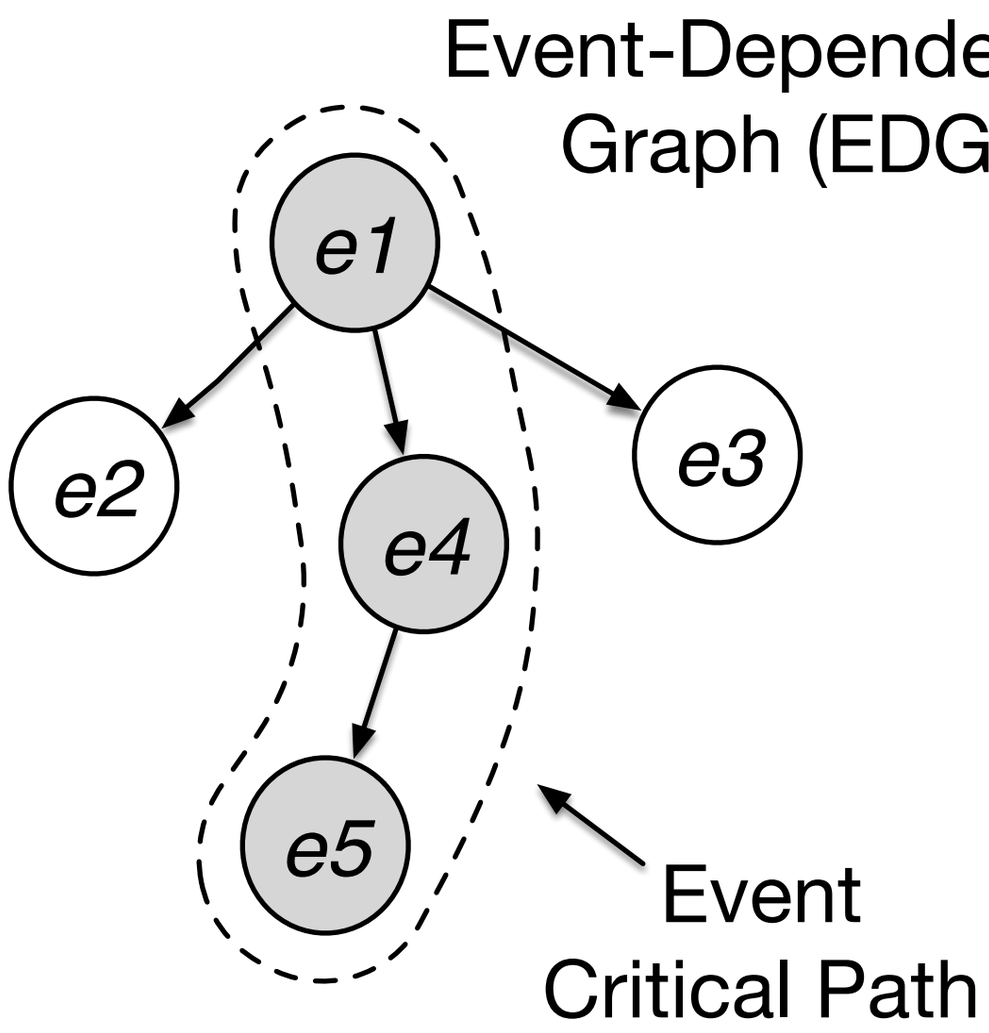
Step 1: Latency Reconstruction

```
var count = N;
fs.readdir("/data", function dir(err, files) {
  files.foreach(function file(f, index) {
    var fname = …;
    fs.readFile(fname, function read(err, data) {
      count -= 1;
      if (count == 0)
        sendResponse();
    });
  });
});
```

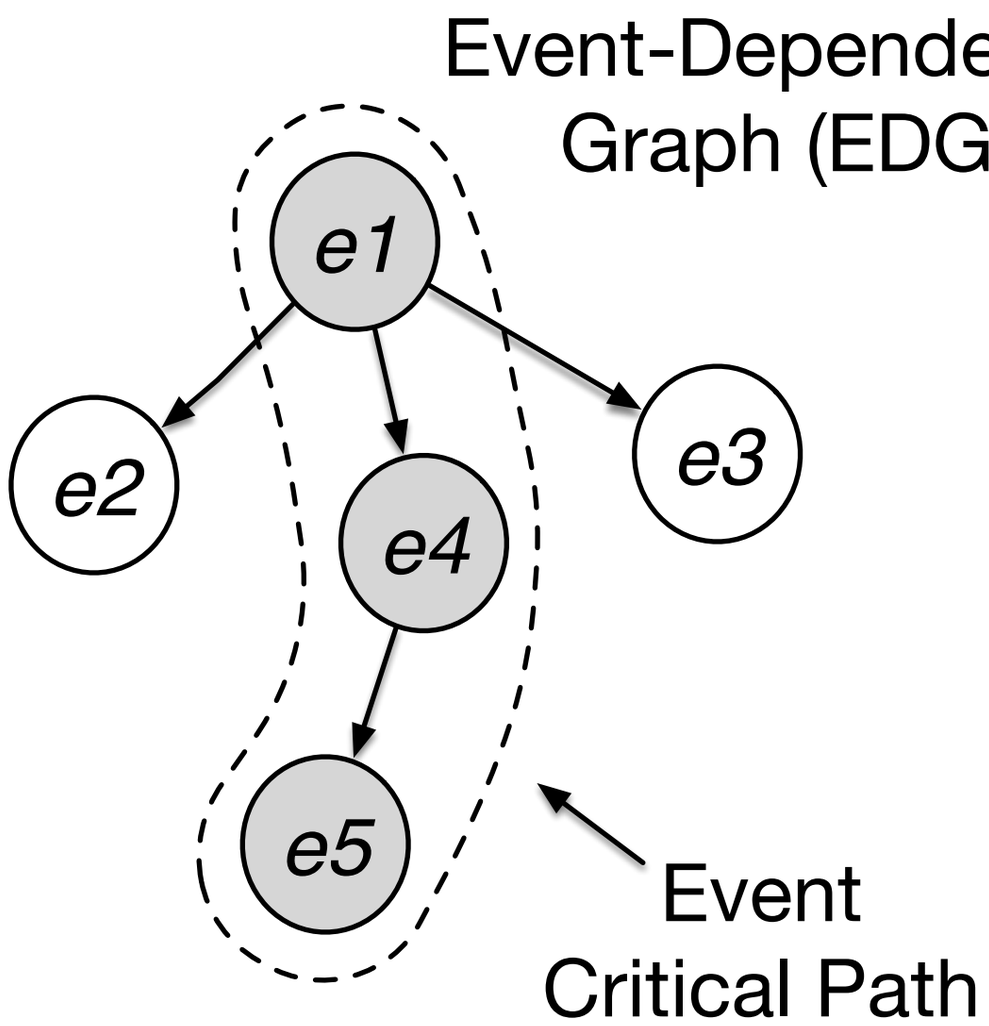


Event Dependency Graph (EDG)

Event Dependency Graph (EDG)

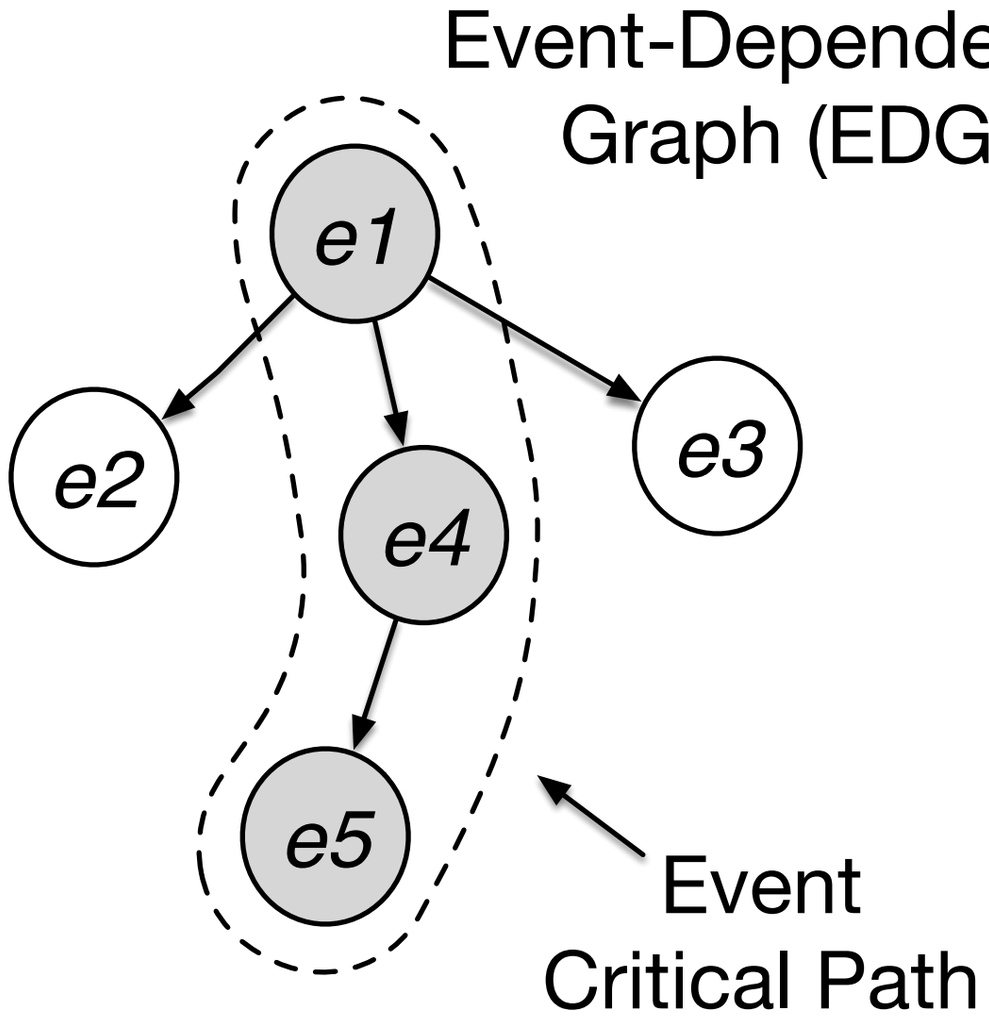


Event Dependency Graph (EDG)



$$Latency(R) = \sum_{i=1}^{N-1} T(e_i), e_i \in ECP(R)$$

Event Dependency Graph (EDG)



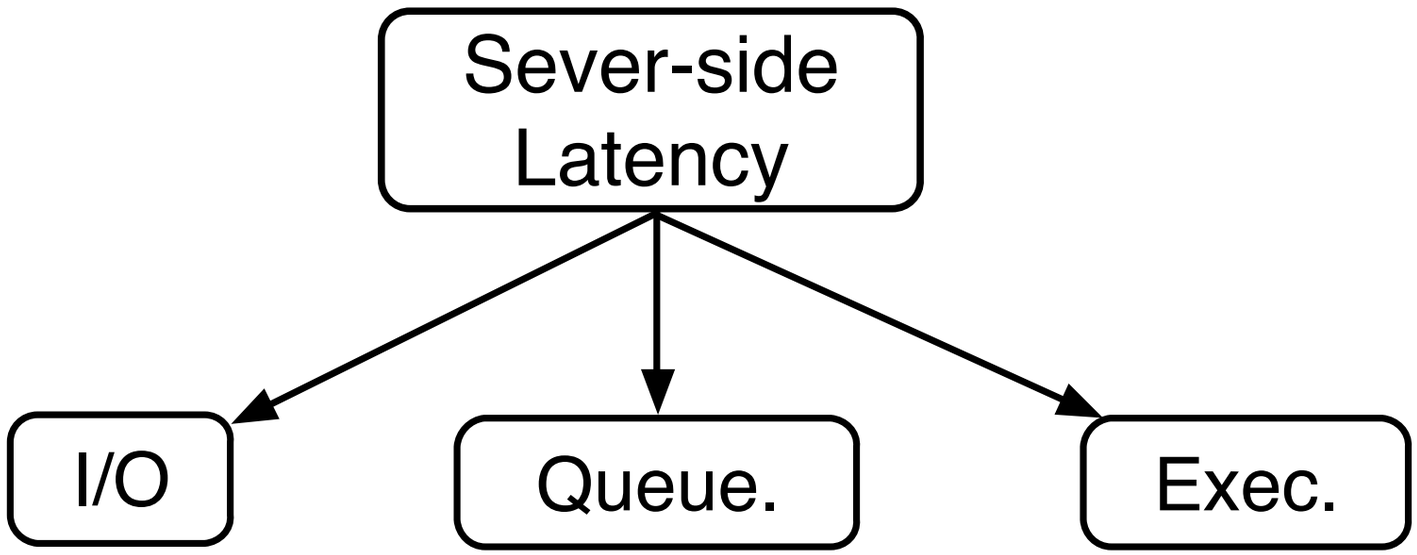
$$Latency(R) = \sum_{i=1}^{N-1} T(e_i), e_i \in ECP(R)$$

How do we obtain the latency of each event?

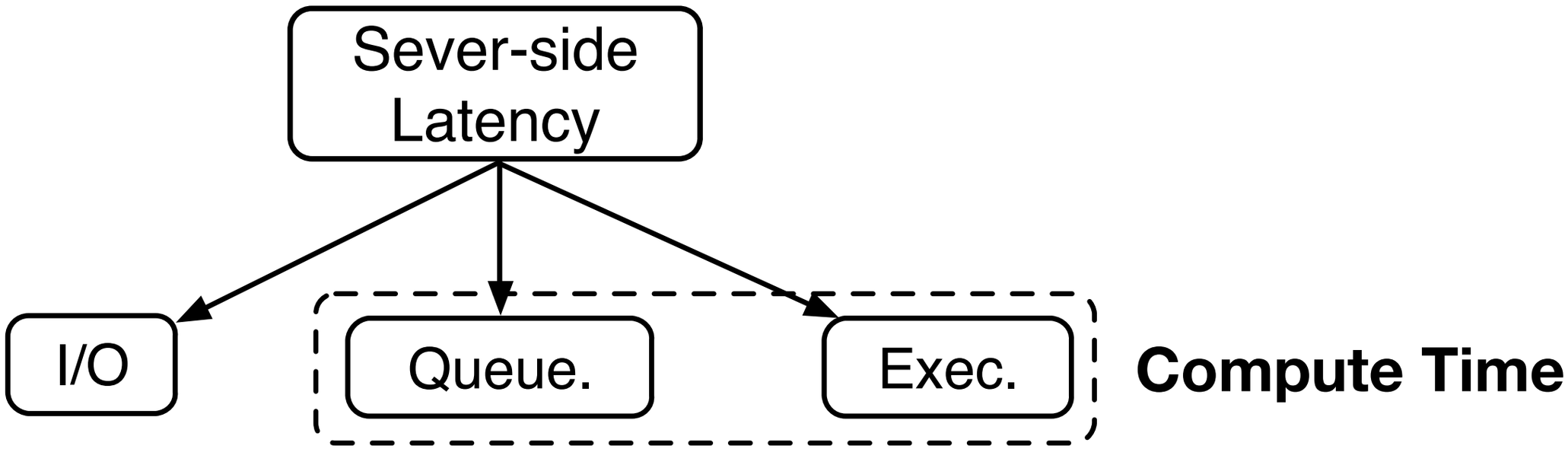
Deconstructing Event Latency

Sever-side Latency

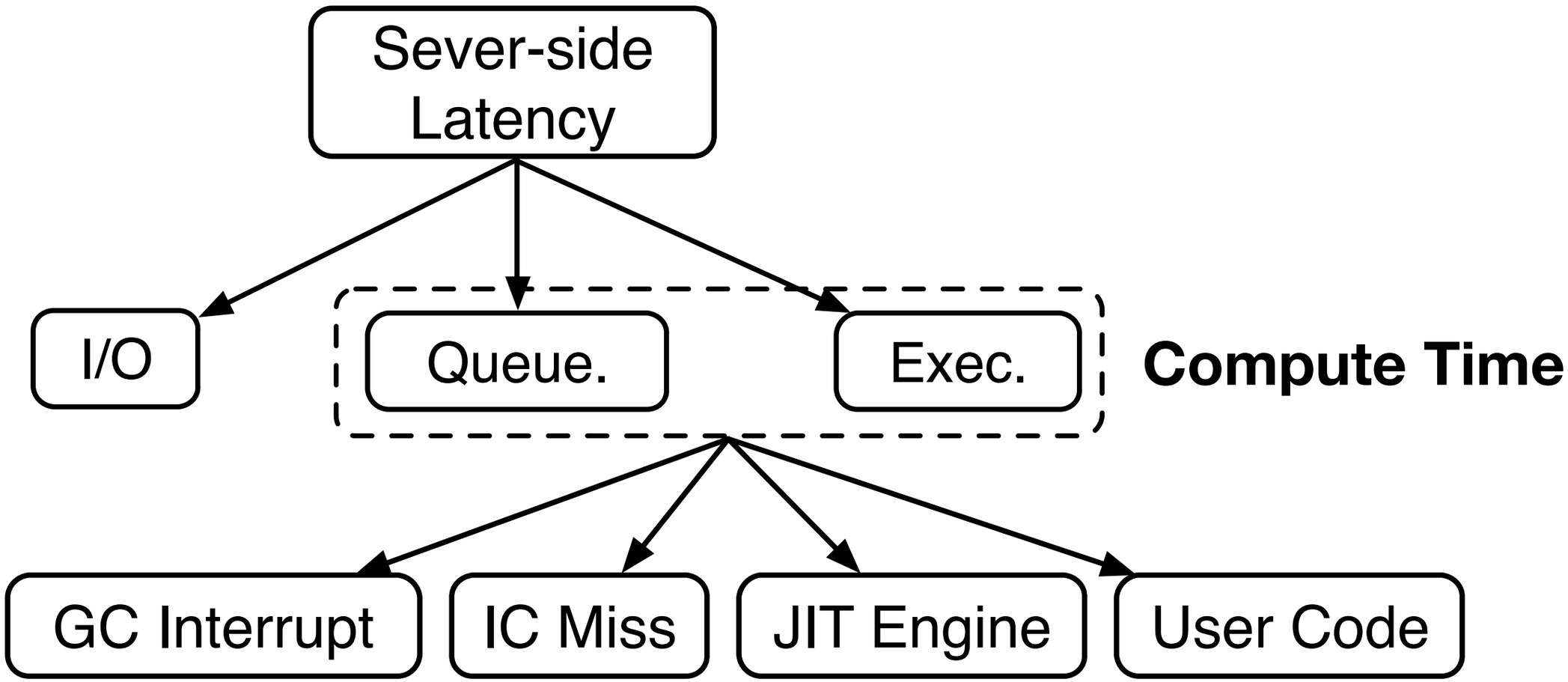
Deconstructing Event Latency



Deconstructing Event Latency

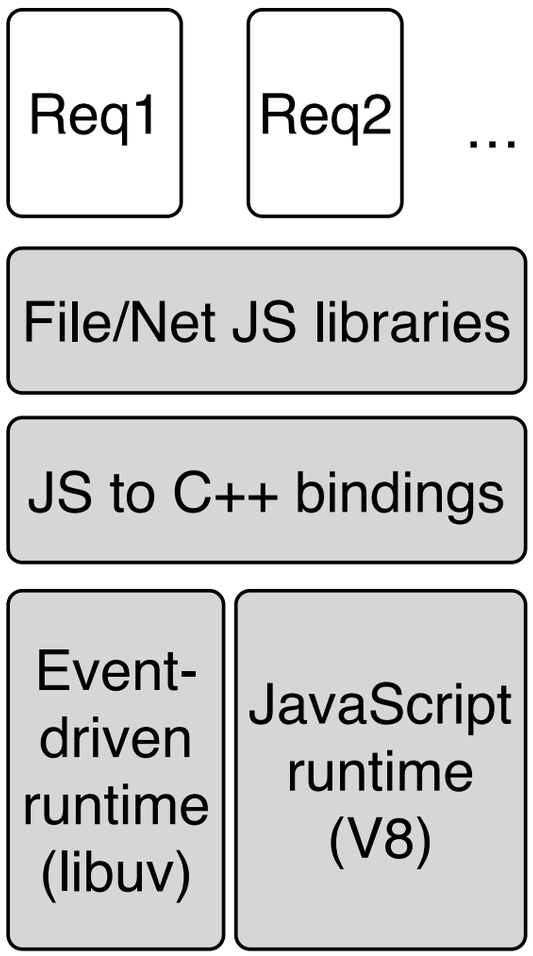


Deconstructing Event Latency



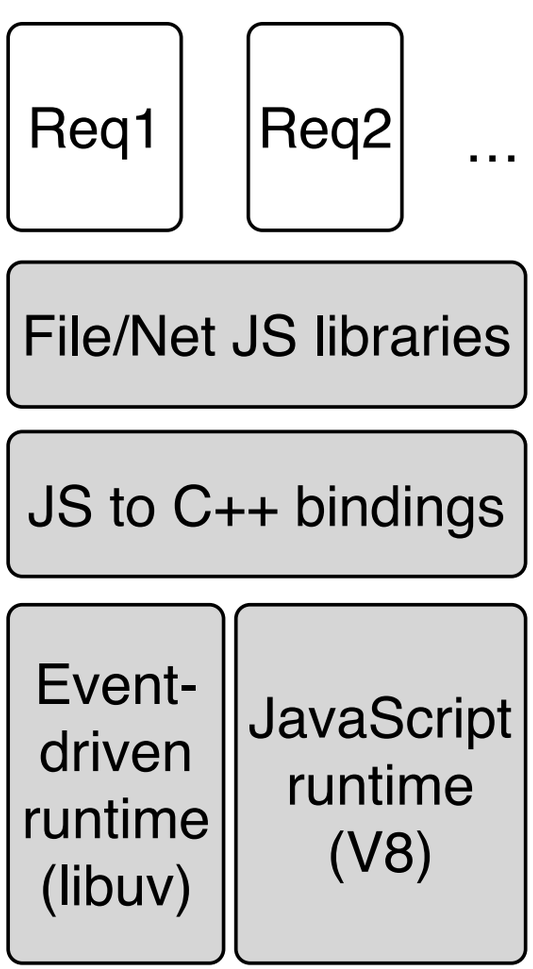
Offline Instrumentation + Online Analysis

Instrument the Node.js runtime so that at runtime we could easily obtain: EDG & event latency info.

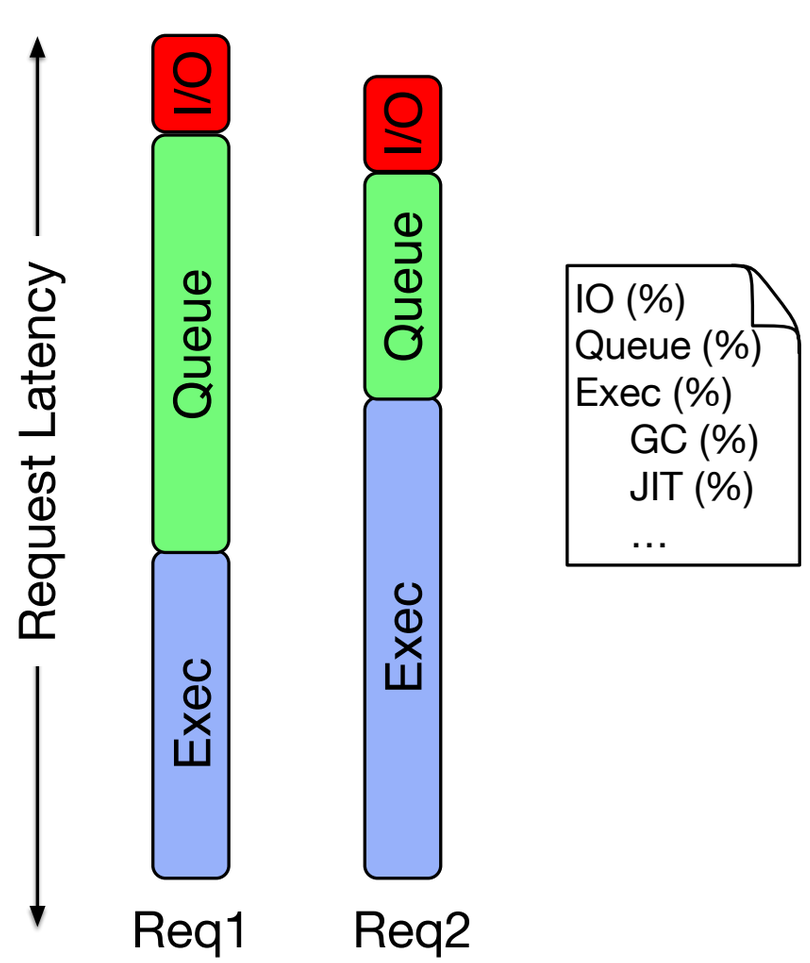


Offline Instrumentation + Online Analysis

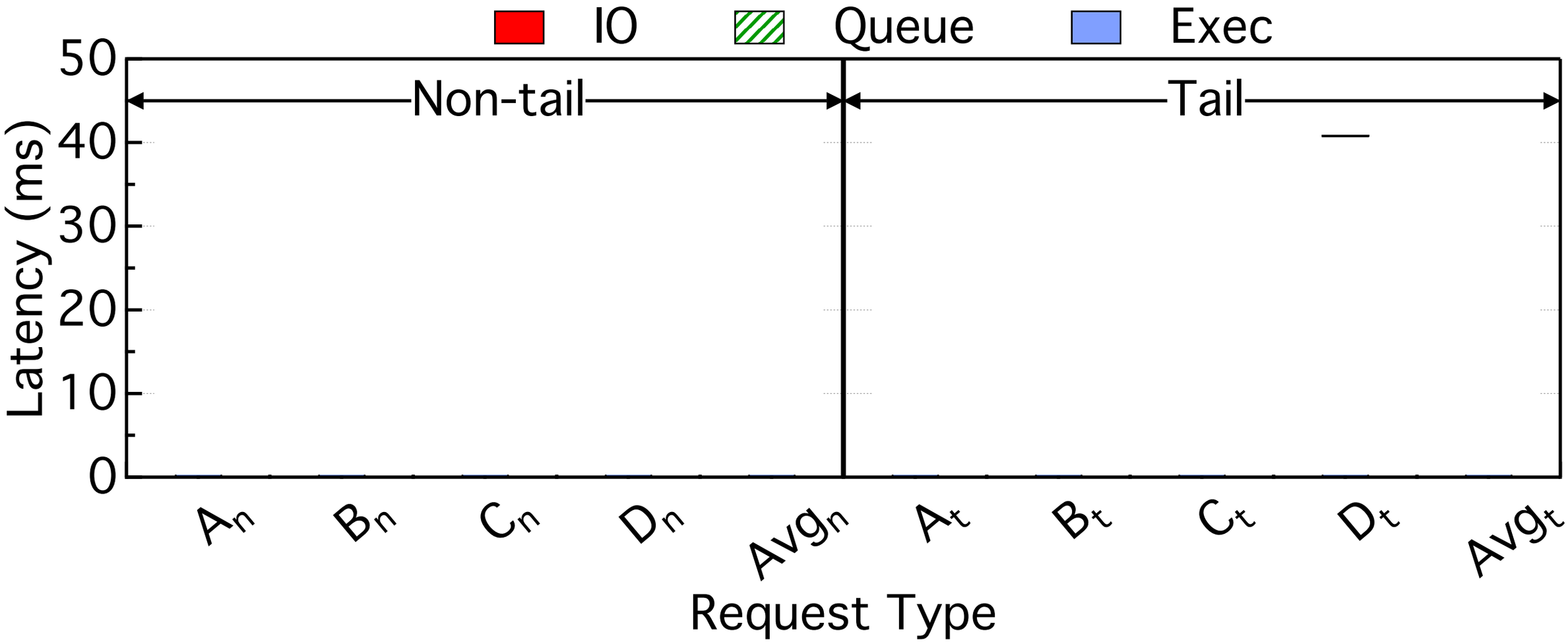
Instrument the Node.js runtime so that at runtime we could easily obtain: EDG & event latency info.



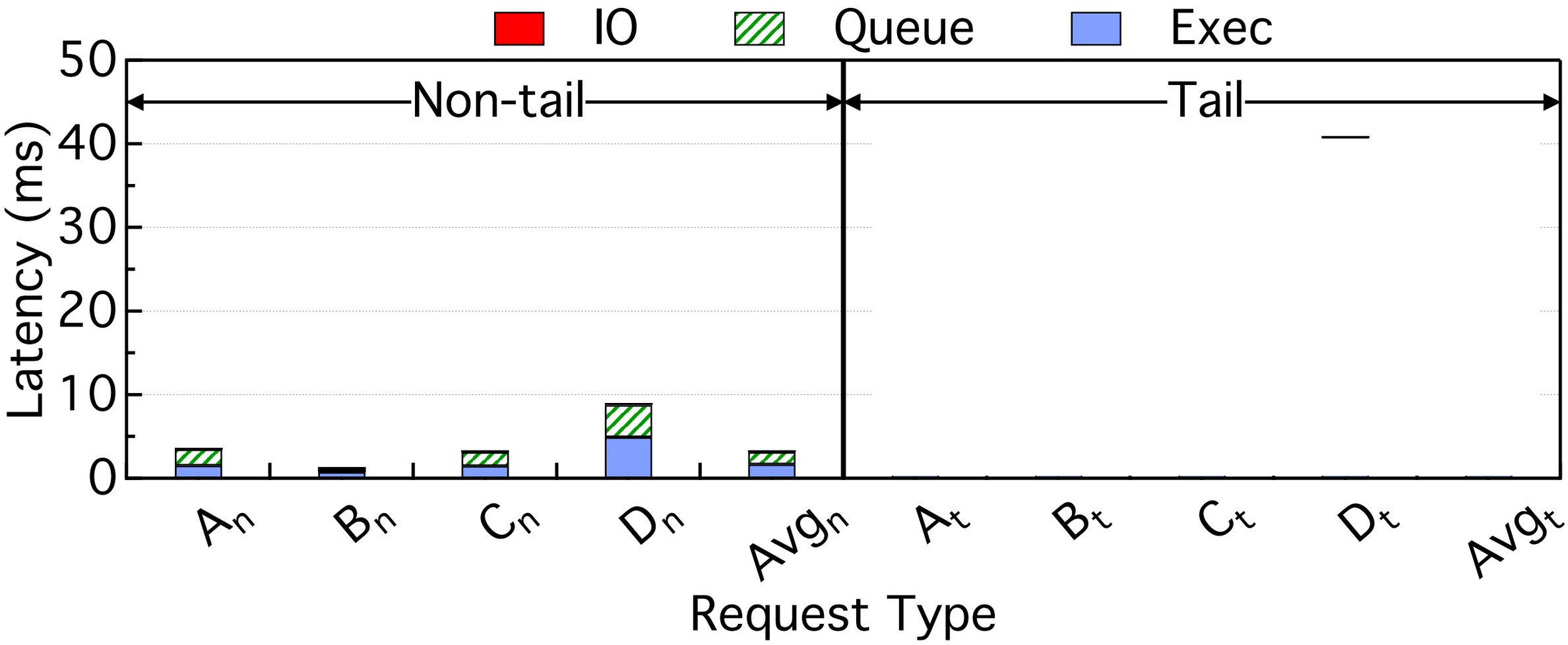
Identify root-causes of long tails at runtime.



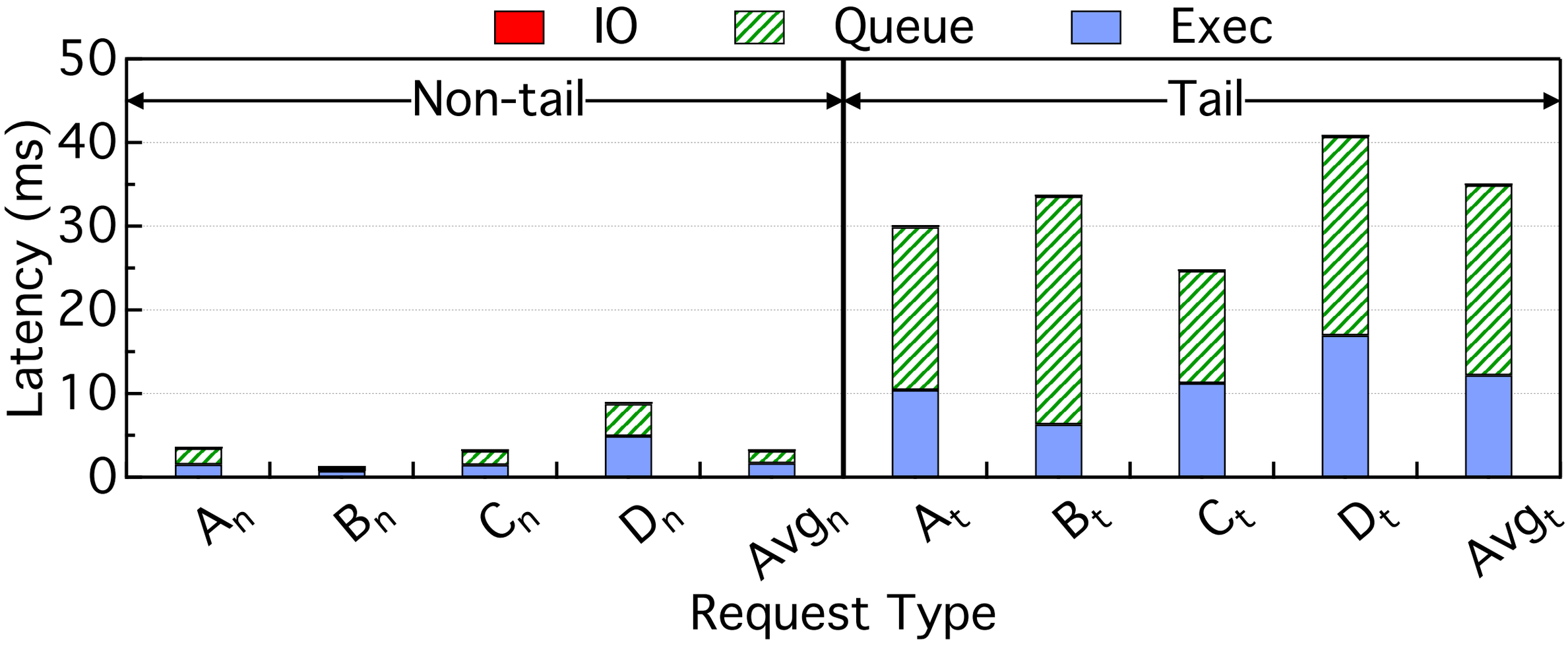
Step 2: EDG-based Bottleneck Analysis



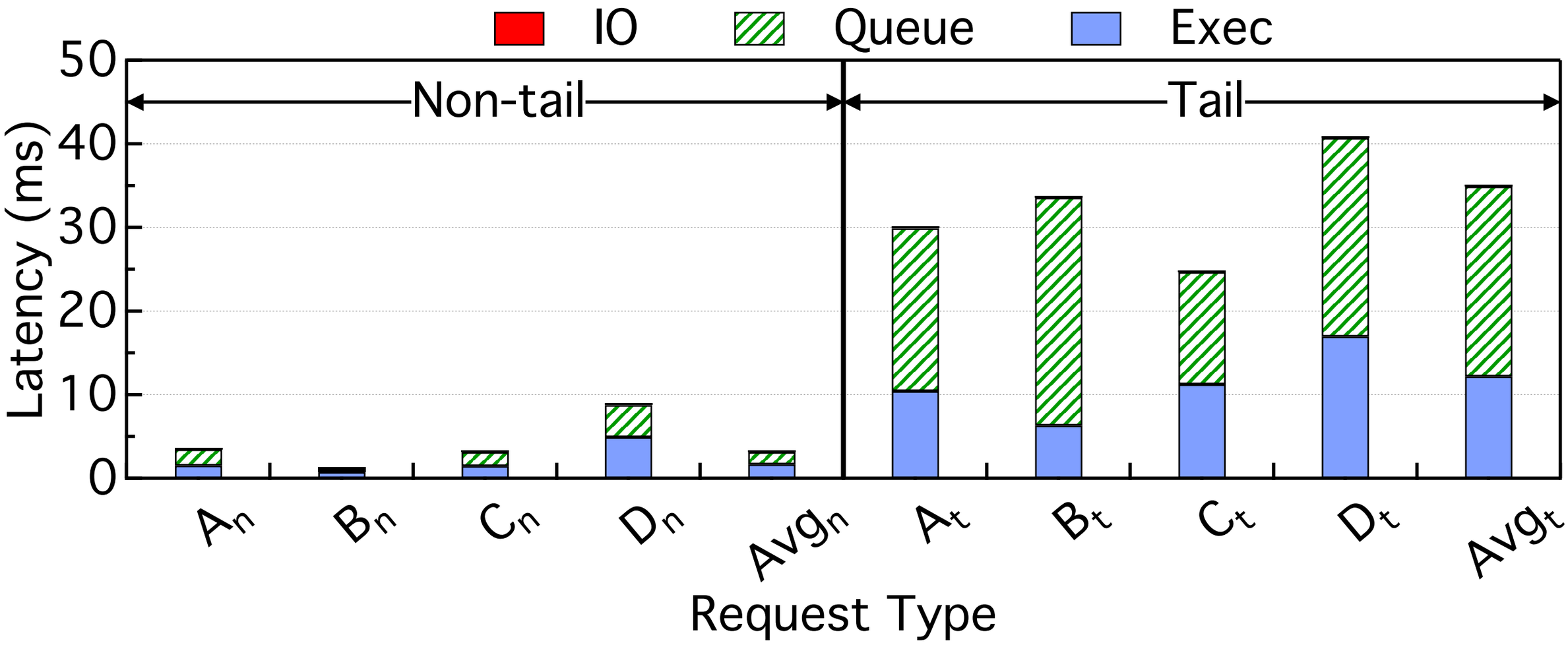
Step 2: EDG-based Bottleneck Analysis



Step 2: EDG-based Bottleneck Analysis

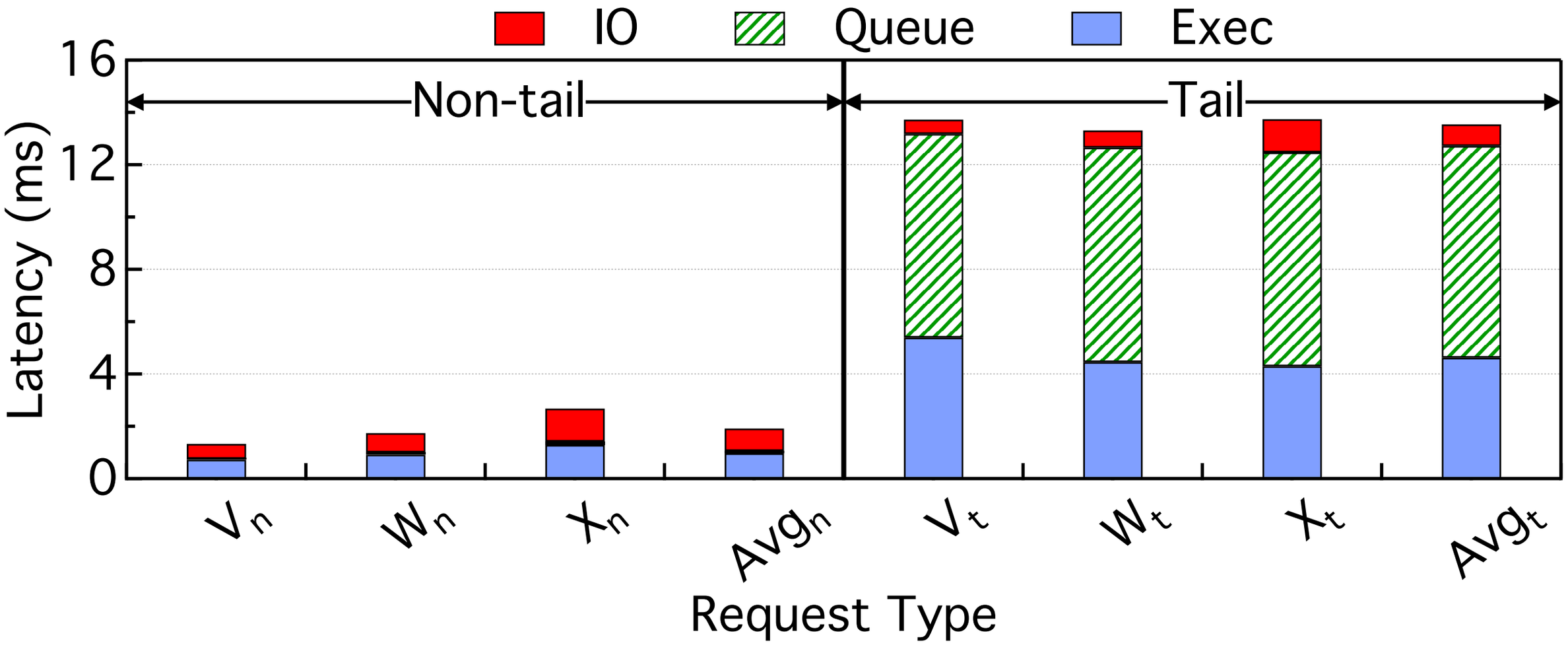


Step 2: EDG-based Bottleneck Analysis

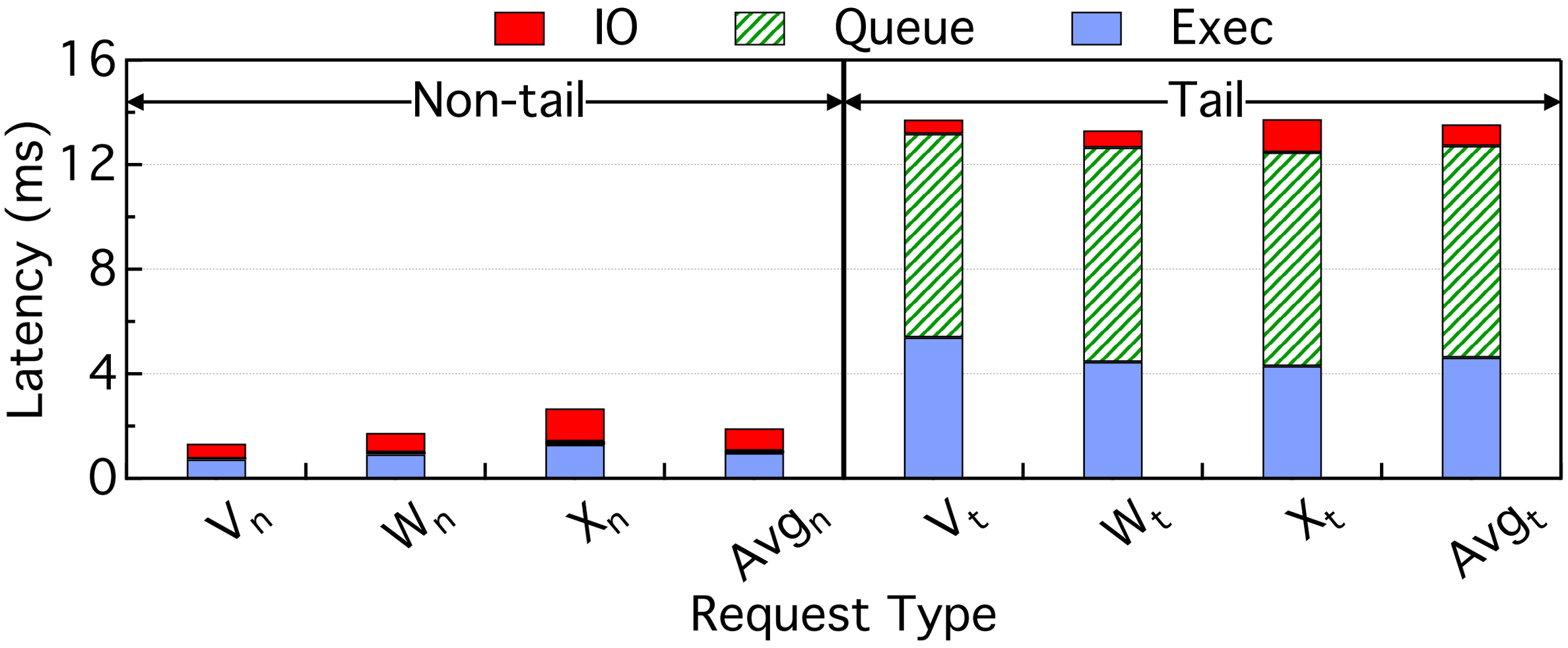


Etherpad: *Queue and Exec. latency increases in tails, and I/O latency is not dominant for this particular application.*

EDG-based Bottleneck Analysis

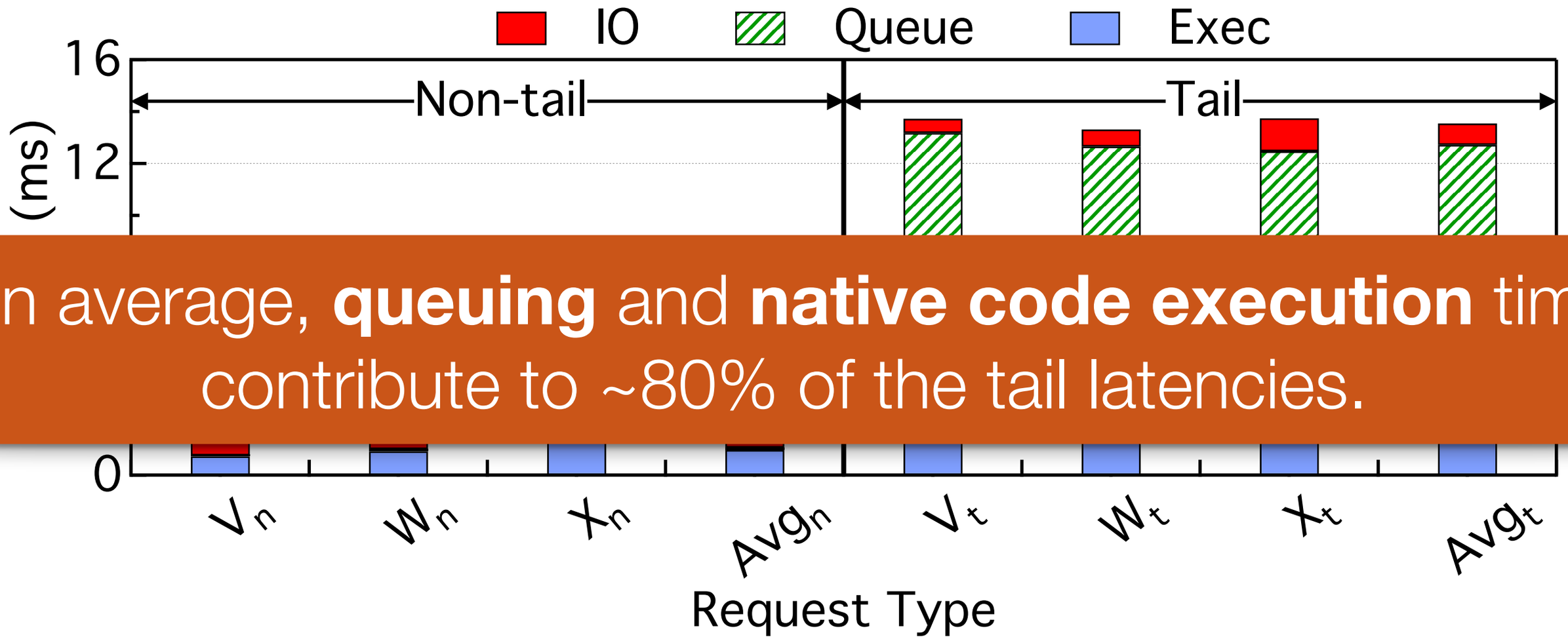


EDG-based Bottleneck Analysis



Client Manager: *Queue and Exec. latency dominate in tails, but unlike Etherpad I/O plays a notable role in the requests.*

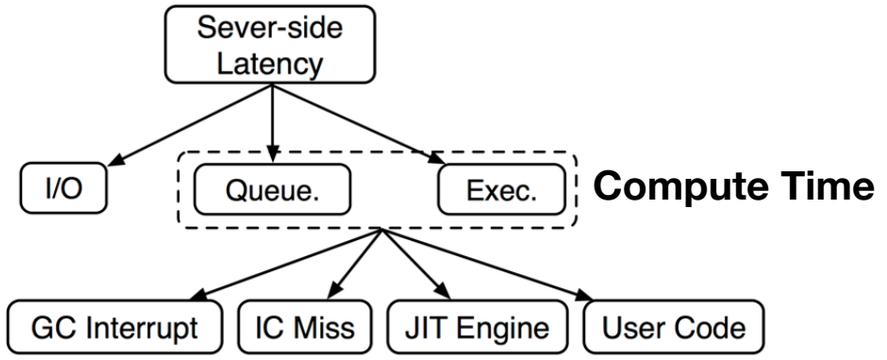
EDG-based Bottleneck Analysis



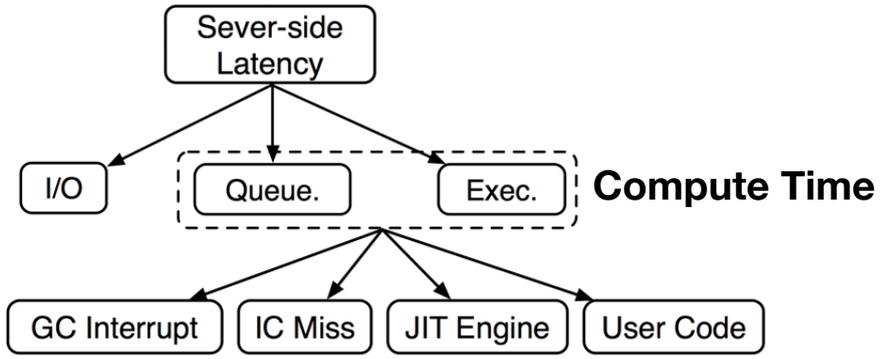
On average, **queuing** and **native code execution** time contribute to ~80% of the tail latencies.

Client Manager: *Queue and Exec. latency dominate in tails, but unlike Etherpad I/O plays a notable role in the requests.*

Breakdown Within Compute

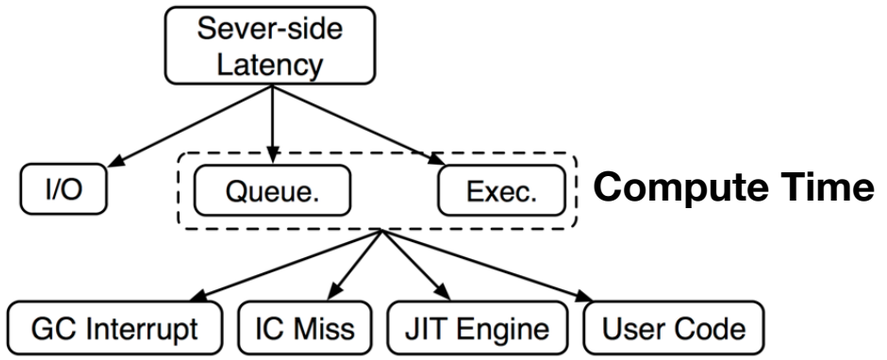


Breakdown Within Compute



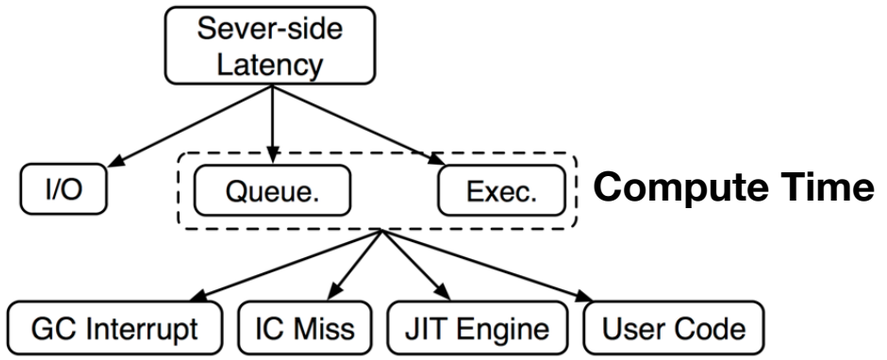
	Non-tail			
Application	**IC**	**GC**	**JIT**	**Native**
Etherpad Lite	0.3%	1.9%	0%	97.8%
Todo	1.8%	0.5%	0%	97.7%
Lighter	4.9%	4.7%	0%	90.4%
Let's Chat	2.7%	3.9%	0%	93.4%
Client Manager	3.2%	3.1%	0%	93.7%

Breakdown Within Compute



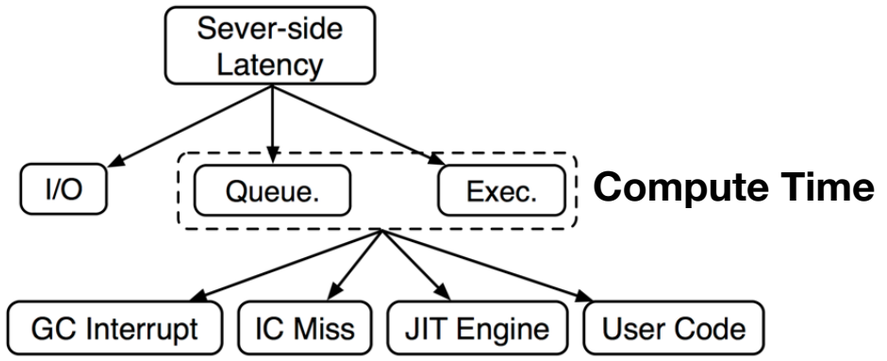
Application	IC	GC	JIT	Non-tail Native
Etherpad Lite	0.3%	1.9%	0%	97.8%
Todo	1.8%	0.5%	0%	97.7%
Lighter	4.9%	4.7%	0%	90.4%
Let's Chat	2.7%	3.9%	0%	93.4%
Client Manager	3.2%	3.1%	0%	93.7%

Breakdown Within Compute



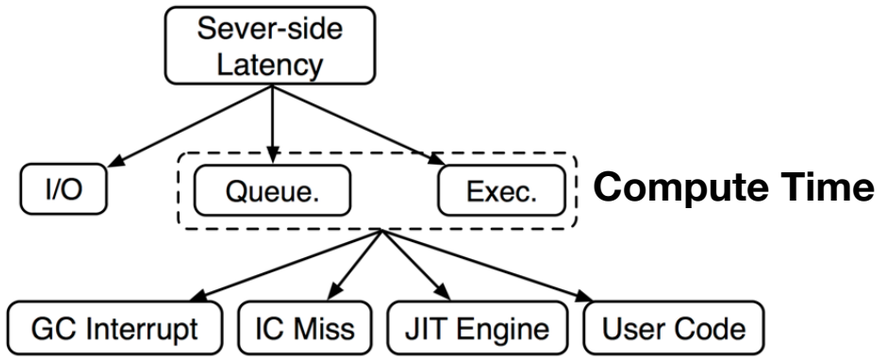
	Non-tail				Tail			
Application	**IC**	**GC**	**JIT**	**Native**	**IC**	**GC**	**JIT**	**Native**
Etherpad Lite	0.3%	1.9%	0%	97.8%	7.0%	41.9%	0%	51.1%
Todo	1.8%	0.5%	0%	97.7%	0.6%	31.0%	0%	68.4%
Lighter	4.9%	4.7%	0%	90.4%	4.4%	45.3%	0%	50.3%
Let's Chat	2.7%	3.9%	0%	93.4%	3.6%	48.4%	0%	48.0%
Client Manager	3.2%	3.1%	0%	93.7%	3.9%	70.3%	0%	25.8%

Breakdown Within Compute



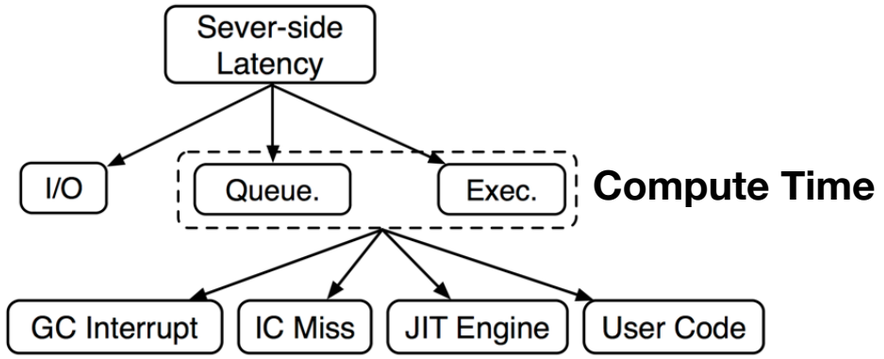
	Non-tail				Tail			
Application	**IC**	**GC**	**JIT**	**Native**	**IC**	**GC**	**JIT**	**Native**
Etherpad Lite	0.3%	1.9%	0%	97.8%	7.0%	41.9%	0%	51.1%
Todo	1.8%	0.5%	0%	97.7%	0.6%	31.0%	0%	68.4%
Lighter	4.9%	4.7%	0%	90.4%	4.4%	45.3%	0%	50.3%
Let's Chat	2.7%	3.9%	0%	93.4%	3.6%	48.4%	0%	48.0%
Client Manager	3.2%	3.1%	0%	93.7%	3.9%	70.3%	0%	25.8%

Breakdown Within Compute



Application	Non-tail IC	Non-tail GC	Non-tail JIT	Non-tail Native	Tail IC	Tail GC	Tail JIT	Tail Native
Etherpad Lite	0.3%	1.9%	0%	97.8%	7.0%	41.9%	0%	51.1%
Todo	1.8%	0.5%	0%	97.7%	0.6%	31.0%	0%	68.4%
Lighter	4.9%	4.7%	0%	90.4%	4.4%	45.3%	0%	50.3%
Let's Chat	2.7%	3.9%	0%	93.4%	3.6%	48.4%	0%	48.0%
Client Manager	3.2%	3.1%	0%	93.7%	3.9%	70.3%	0%	25.8%

Breakdown Within Compute

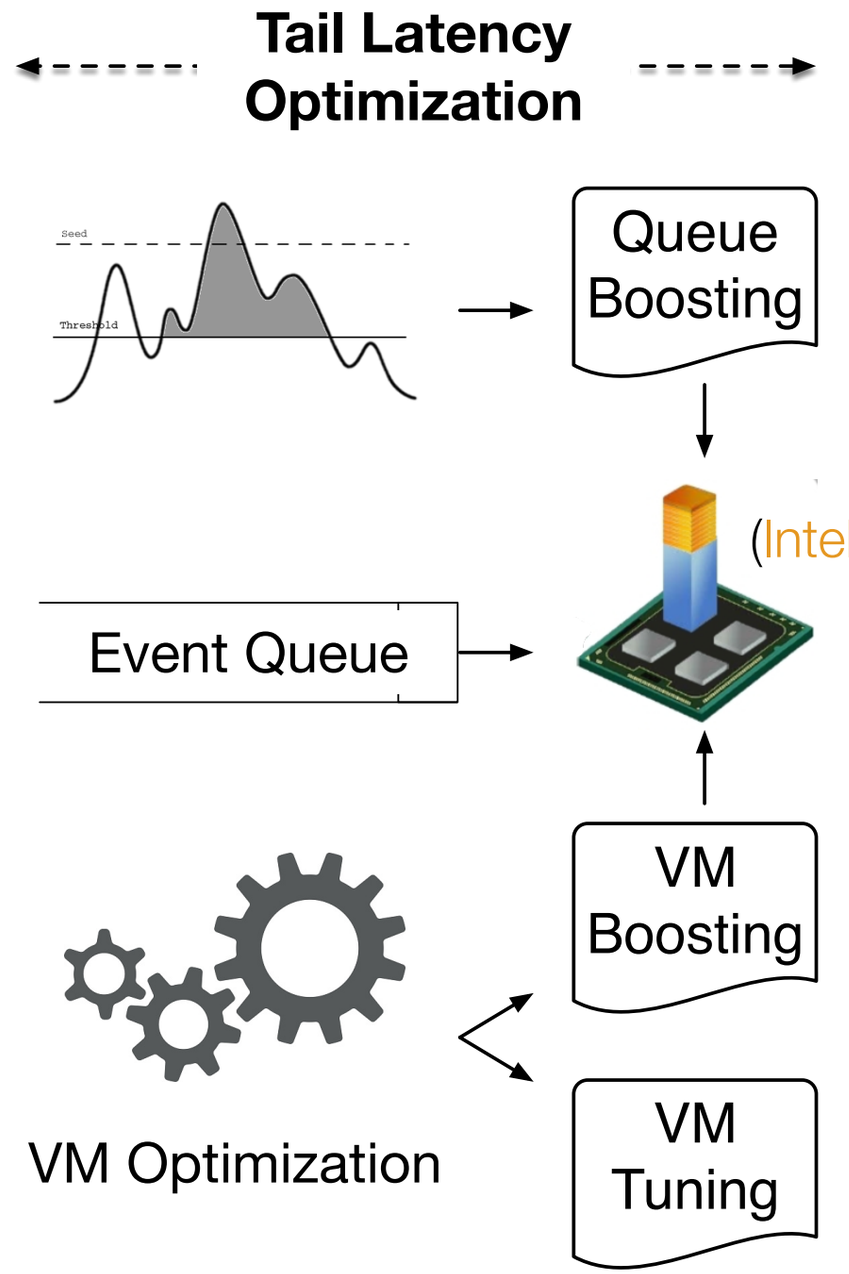


	Non-tail				Tail			
Application	IC	GC	JIT	Native	IC	GC	JIT	Native
Let's Chat	2.7%	3.9%	0%	93.4%	3.6%	48.4%	0%	48.0%
Client Manager	3.2%	3.1%	0%	93.7%	3.9%	70.3%	0%	25.8%

We should focus optimization efforts on **Garbage Collection** and **Generated Native Code**

Step 3: Tail Latency Optimization

- Leveraging the **turbo boosting** capability of modern CPUs
- Key: wisely choose what to boost
- GC Boosting
  - Boost GC
- Queue Boosting
  - Boost when the system is “busy”
  - Use event queue stats as “hints”



Optimization 1: VM Optimization (GC Boost)

	Non-tail				Tail			
Application	**IC**	**GC**	**JIT**	**Native**	**IC**	**GC**	**JIT**	**Native**
Etherpad Lite	0.3%	1.9%	0%	97.8%	7.0%	41.9%	0%	51.1%
Todo	1.8%	0.5%	0%	97.7%	0.6%	31.0%	0%	68.4%
Lighter	4.9%	4.7%	0%	90.4%	4.4%	45.3%	0%	50.3%
Let's Chat	2.7%	3.9%	0%	93.4%	3.6%	48.4%	0%	48.0%
Client Manager	3.2%	3.1%	0%	93.7%	3.9%	70.3%	0%	25.8%

Optimization 1: VM Optimization (GC Boost)

- **Observations:**
  - GCs are infrequent, little overall energy overhead
  - IPC during GC is relatively high: ~1.3

	Non-tail				Tail			
Application	**IC**	**GC**	**JIT**	**Native**	**IC**	**GC**	**JIT**	**Native**
Etherpad Lite	0.3%	1.9%	0%	97.8%	7.0%	41.9%	0%	51.1%
Todo	1.8%	0.5%	0%	97.7%	0.6%	31.0%	0%	68.4%
Lighter	4.9%	4.7%	0%	90.4%	4.4%	45.3%	0%	50.3%
Let's Chat	2.7%	3.9%	0%	93.4%	3.6%	48.4%	0%	48.0%
Client Manager	3.2%	3.1%	0%	93.7%	3.9%	70.3%	0%	25.8%

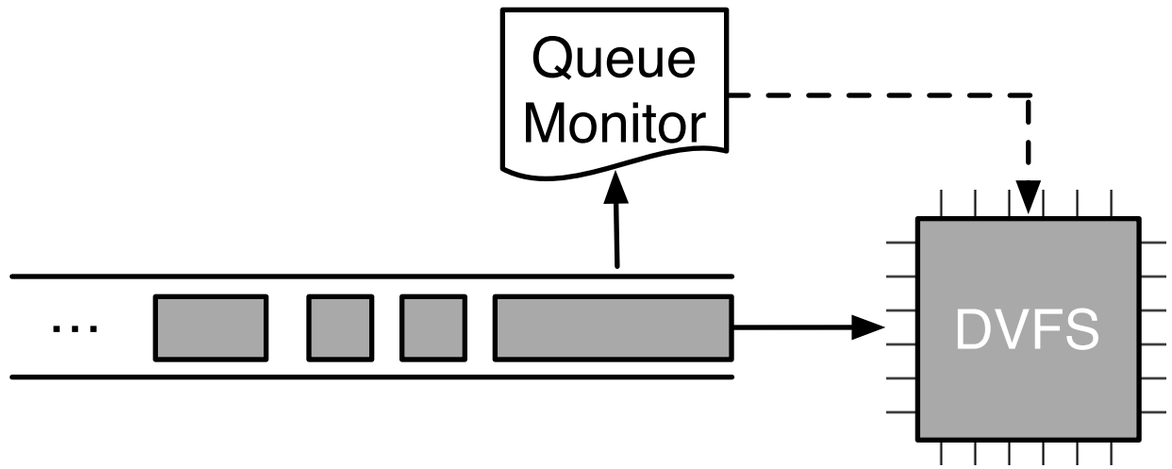
Optimization 1: VM Optimization (GC Boost)

- **Observations:**
  - GCs are infrequent, little overall energy overhead
  - IPC during GC is relatively high: ~1.3

	Non-tail				Tail			
Application	**IC**	**GC**	**JIT**	**Native**	**IC**	**GC**	**JIT**	**Native**
Etherpad Lite	0.3%	1.9%	0%	97.8%	7.0%	41.9%	0%	51.1%
Todo	1.8%	0.5%	0%	97.7%	0.6%	31.0%	0%	68.4%
Lighter	4.9%	4.7%	0%	90.4%	4.4%	45.3%	0%	50.3%
Let's Chat	2.7%	3.9%	0%	93.4%	3.6%	48.4%	0%	48.0%
Client Manager	3.2%	3.1%	0%	93.7%	3.9%	70.3%	0%	25.8%

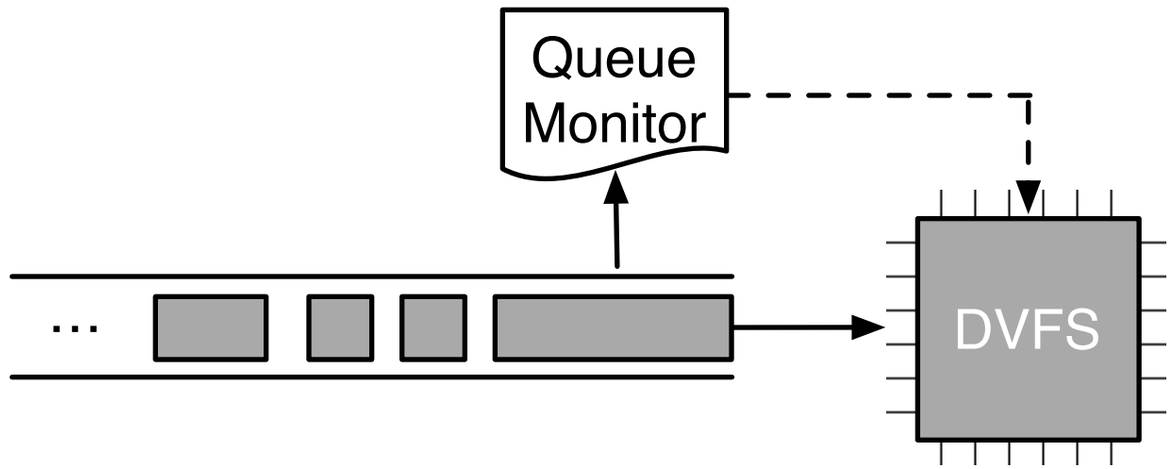
- **Implementation**
  - User-space DVFS embedded in Google V8: enter boosting at GC **prologues** and exit boosting at GC **epilogues**
  - More benefits if we have access to fine-grained per-core DVFS mechanism

Optimization 2: Queue Boost



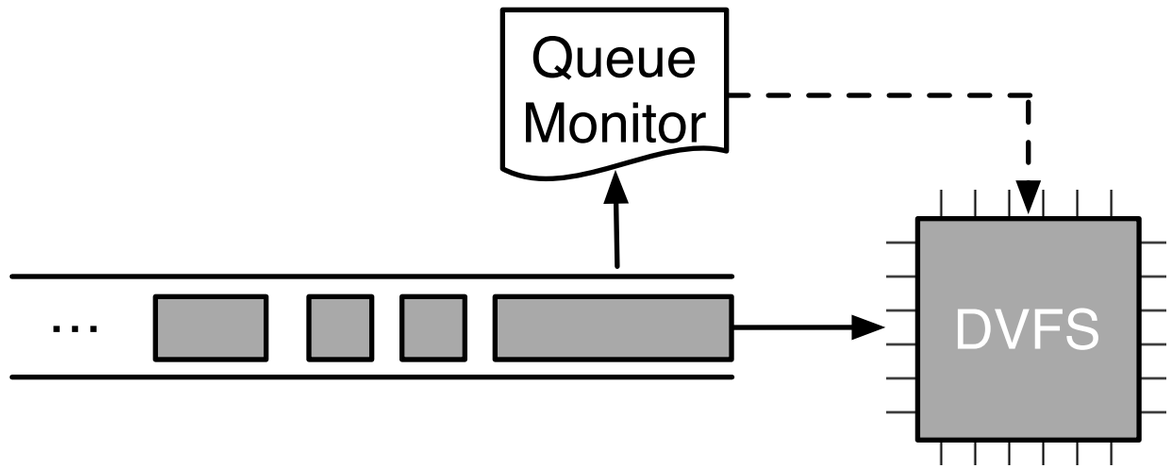
Optimization 2: Queue Boost

- More general compute acceleration: Boost when the system is “busy”



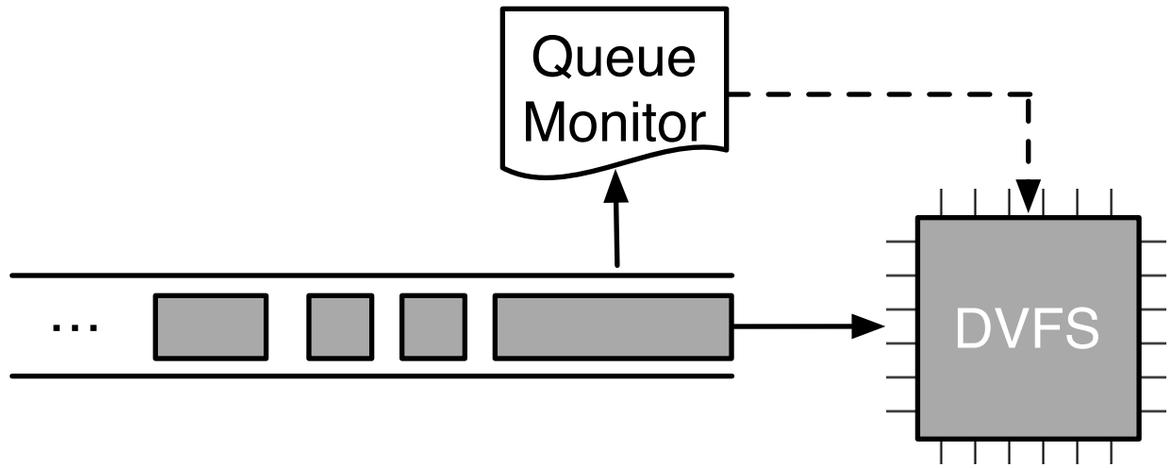
Optimization 2: Queue Boost

- More general compute acceleration: Boost when the system is “busy”
- How do you detect that?



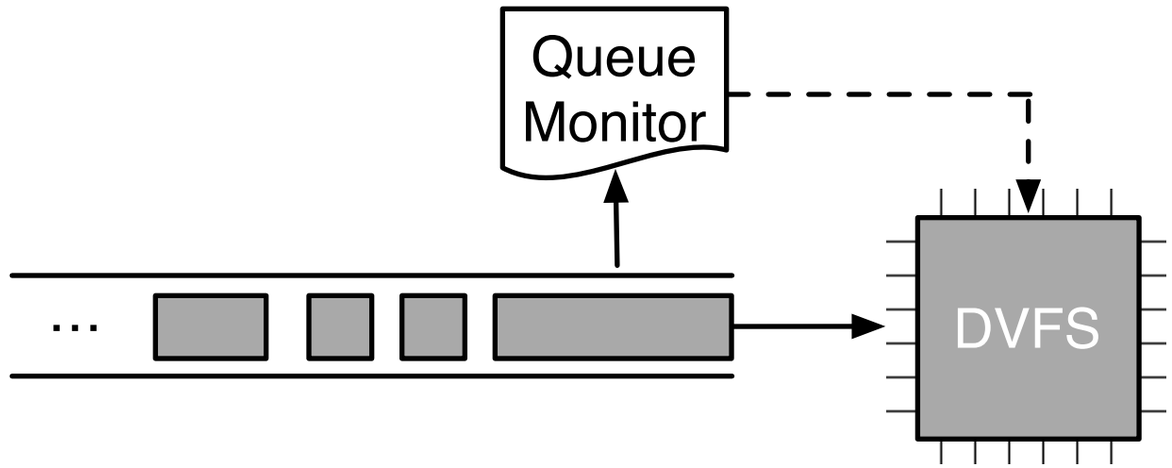
Optimization 2: Queue Boost

- More general compute acceleration: Boost when the system is “busy”
- How do you detect that?
- Rely on two queue-related heuristics:
  - # of events in the queue
  - Processing time of the head-of-line event



Optimization 2: Queue Boost

- More general compute acceleration: Boost when the system is “busy”
- How do you detect that?
- Rely on two queue-related heuristics:
  - # of events in the queue
  - Processing time of the head-of-line event



- Implementation:
  - Periodic **Sampling:** Every 1 ms
  - Dynamic **Thresholding**
    - Sample the average value of event number and per event processing time
    - Amplify the average value to decide a dynamic threshold by 2-3x

Evaluation

- **Our system:** normal operating frequency at 2.6 GHz, boosts to max 4.0 GHz during GC boost and Queue boost

Evaluation

- **Our system:** normal operating frequency at 2.6 GHz, boosts to max 4.0 GHz during GC boost and Queue boost
- **Different Variants:**

Evaluation

- **Our system:** normal operating frequency at 2.6 GHz, boosts to max 4.0 GHz during GC boost and Queue boost
- **Different Variants:**
  - GC Boost

Evaluation

- **Our system:** normal operating frequency at 2.6 GHz, boosts to max 4.0 GHz during GC boost and Queue boost
- **Different Variants:**
  - GC Boost
  - GC Boost with GC Parameter Tuning

Evaluation

- **Our system:** normal operating frequency at 2.6 GHz, boosts to max 4.0 GHz during GC boost and Queue boost
- **Different Variants:**
  - GC Boost
  - GC Boost with GC Parameter Tuning
  - Queue Boost

Evaluation

- **Our system:** normal operating frequency at 2.6 GHz, boosts to max 4.0 GHz during GC boost and Queue boost
- **Different Variants:**
  - GC Boost
  - GC Boost with GC Parameter Tuning
  - Queue Boost
  - GC Tuning + GC Boost + Queue Boost

Evaluation

- **Our system:** normal operating frequency at 2.6 GHz, boosts to max 4.0 GHz during GC boost and Queue boost
- **Different Variants:**
  - GC Boost
  - GC Boost with GC Parameter Tuning
  - Queue Boost
  - GC Tuning + GC Boost + Queue Boost
- **Baseline**

Evaluation

- **Our system:** normal operating frequency at 2.6 GHz, boosts to max 4.0 GHz during GC boost and Queue boost
- **Different Variants:**
  - GC Boost
  - GC Boost with GC Parameter Tuning
  - Queue Boost
  - GC Tuning + GC Boost + Queue Boost
- **Baseline**
  - **Static Frequency**: 3.3GHz and 4.0GHz

Evaluation

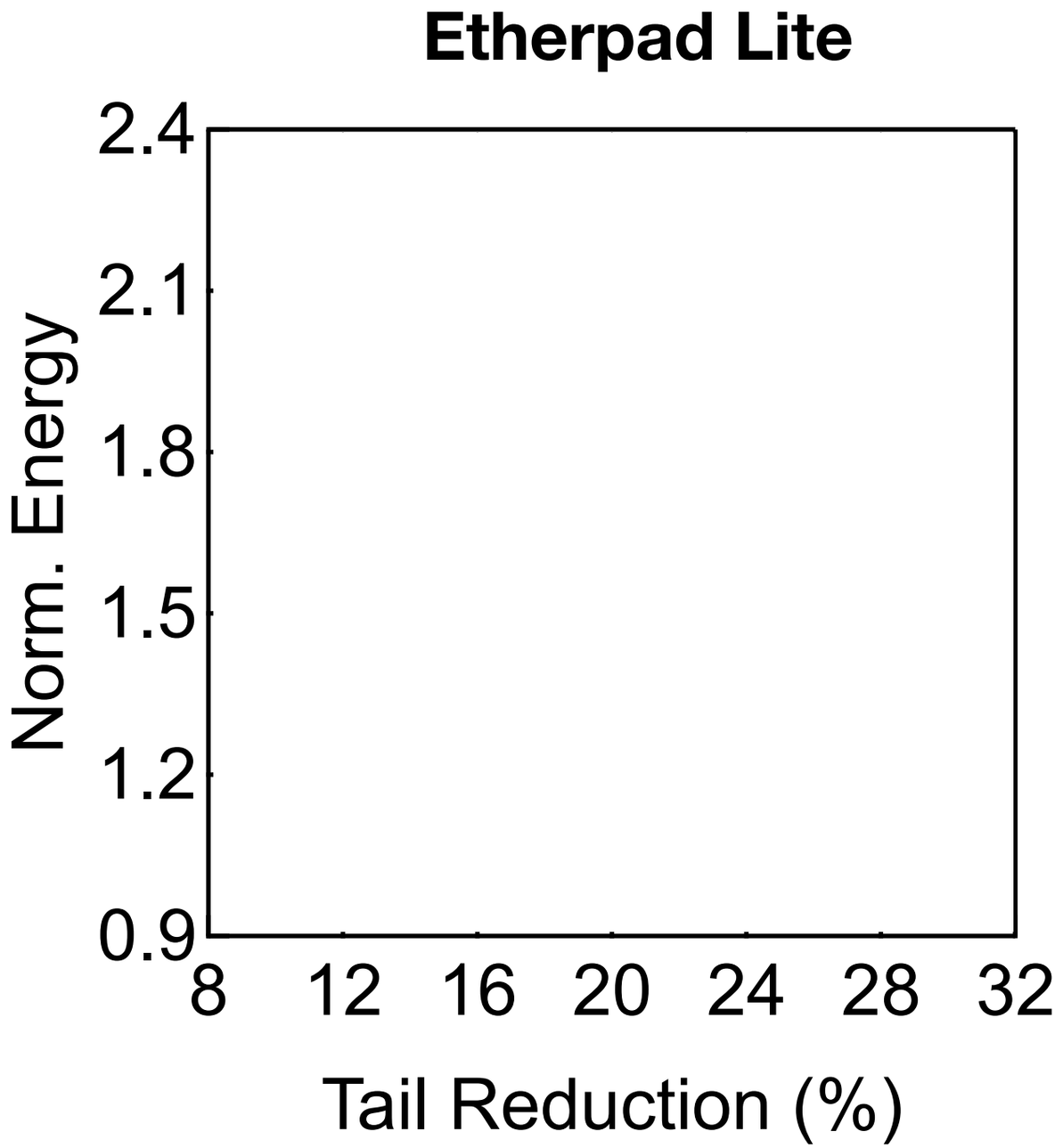
- **Our system:** normal operating frequency at 2.6 GHz, boosts to max 4.0 GHz during GC boost and Queue boost
- **Different Variants:**
  - GC Boost
  - GC Boost with GC Parameter Tuning
  - Queue Boost
  - GC Tuning + GC Boost + Queue Boost
- **Baseline**
  - **Static Frequency**: 3.3GHz and 4.0GHz
  - **Adrenaline** [HPCA 2015]

Evaluation

- **Our system:** normal operating frequency at 2.6 GHz, boosts to max 4.0 GHz during GC boost and Queue boost
- **Different Variants:**
  - GC Boost
  - GC Boost with GC Parameter Tuning
  - Queue Boost
  - GC Tuning + GC Boost + Queue Boost
- **Baseline**
  - **Static Frequency**: 3.3GHz and 4.0GHz
  - **Adrenaline** [HPCA 2015]
  - **Rubik** [MICRO 2015]

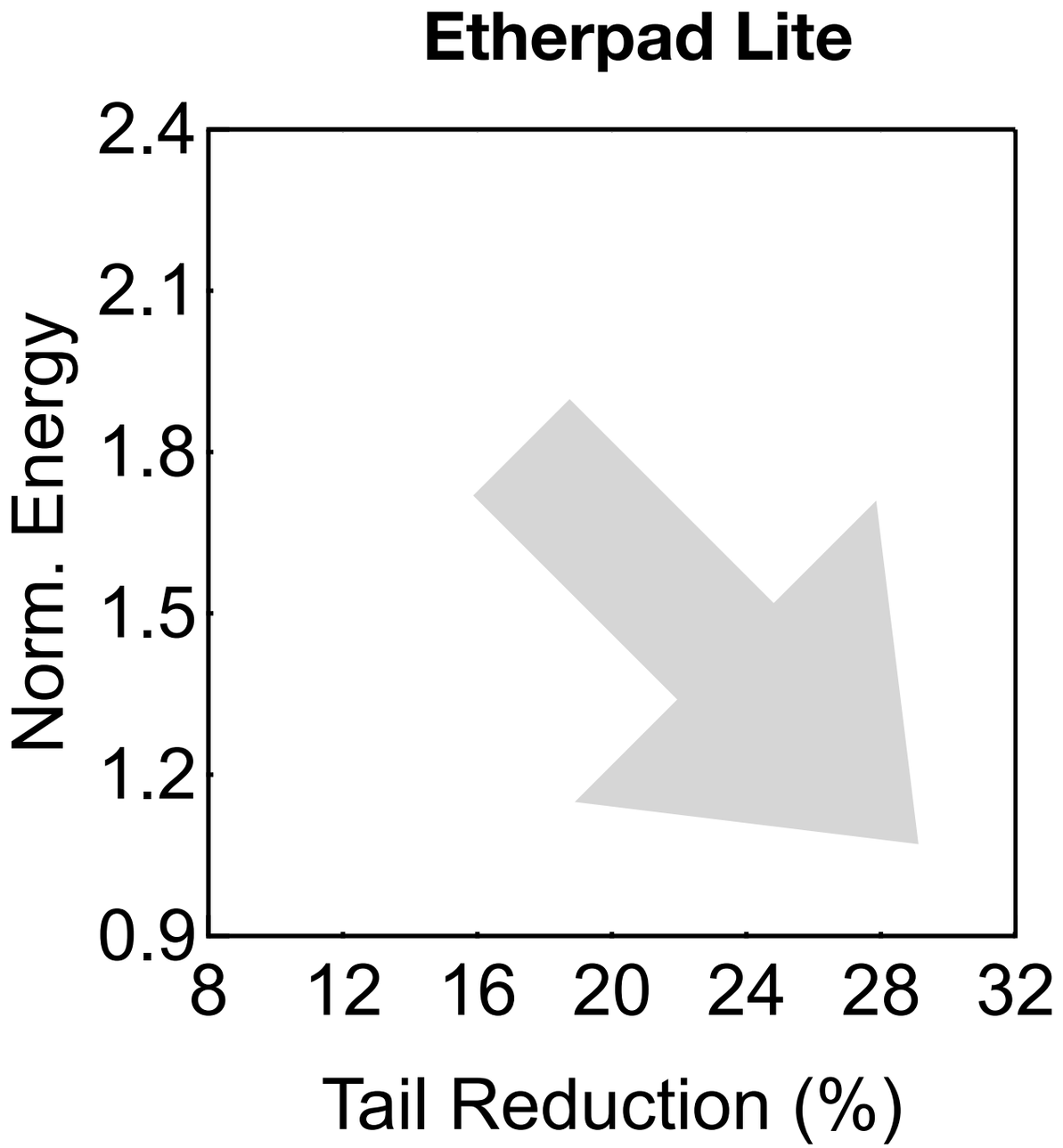
Evaluation

- **Our system:** normal operating frequency at 2.6 GHz, boosts to max 4.0 GHz during GC boost and Queue boost
- **Different Variants:**
  - GC Boost
  - GC Boost with GC Parameter Tuning
  - Queue Boost
  - GC Tuning + GC Boost + Queue Boost
- **Baseline**
  - **Static Frequency**: 3.3GHz and 4.0GHz
  - **Adrenaline** [HPCA 2015]
  - **Rubik** [MICRO 2015]



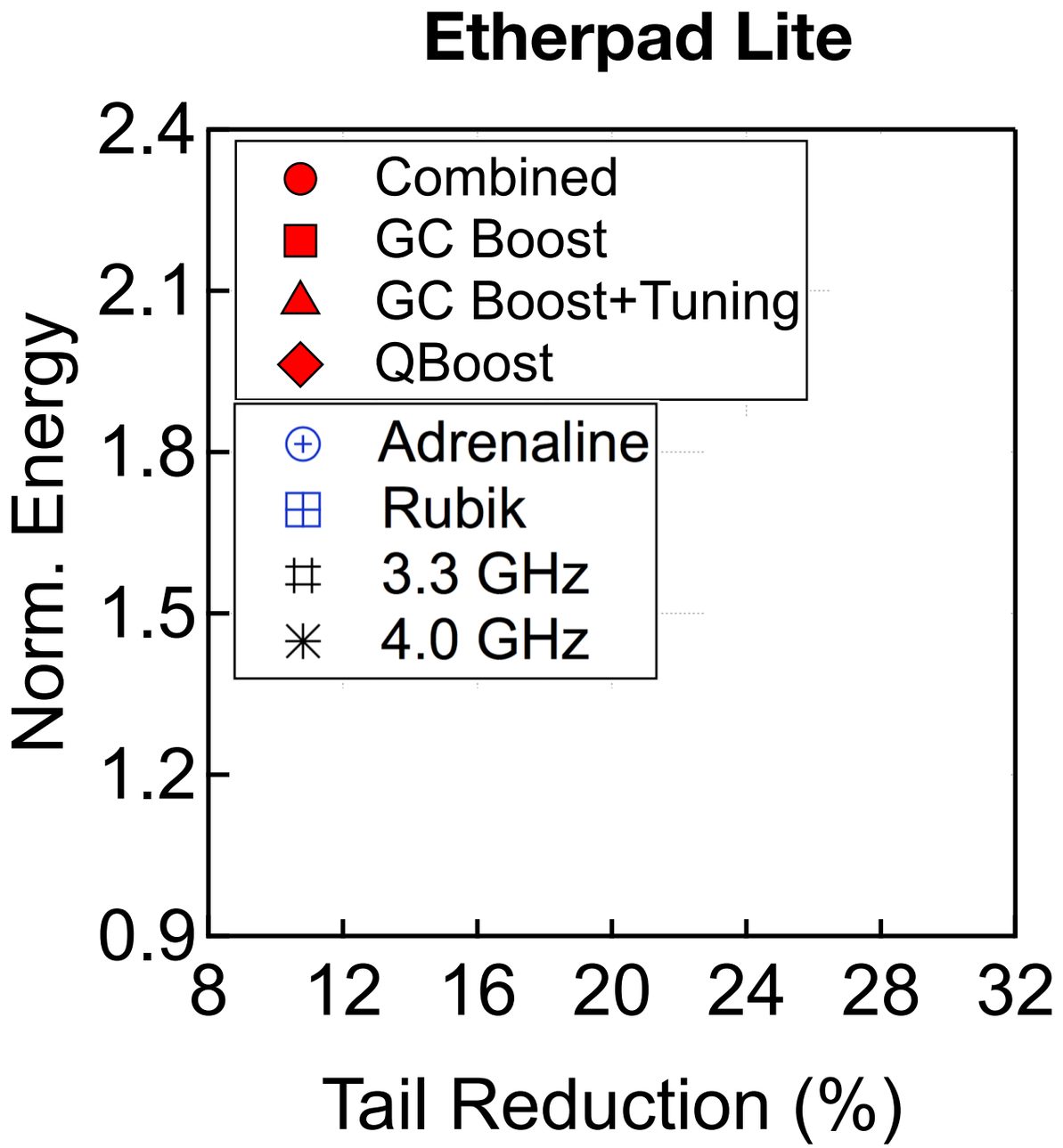
Evaluation

- **Our system:** normal operating frequency at 2.6 GHz, boosts to max 4.0 GHz during GC boost and Queue boost
- **Different Variants:**
  - GC Boost
  - GC Boost with GC Parameter Tuning
  - Queue Boost
  - GC Tuning + GC Boost + Queue Boost
- **Baseline**
  - **Static Frequency**: 3.3GHz and 4.0GHz
  - **Adrenaline** [HPCA 2015]
  - **Rubik** [MICRO 2015]



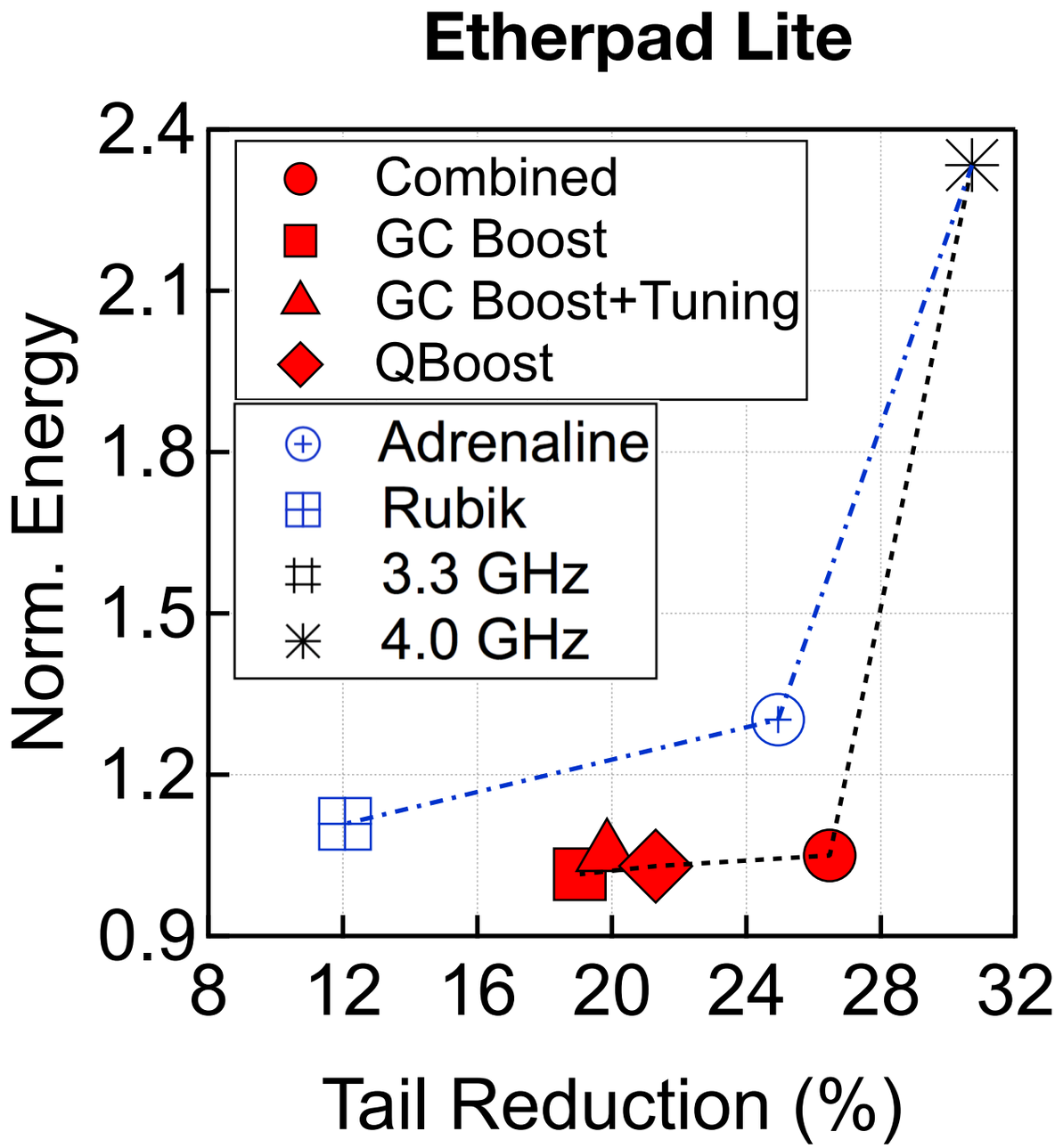
Evaluation

- **Our system:** normal operating frequency at 2.6 GHz, boosts to max 4.0 GHz during GC boost and Queue boost
- **Different Variants:**
  - GC Boost
  - GC Boost with GC Parameter Tuning
  - Queue Boost
  - GC Tuning + GC Boost + Queue Boost
- **Baseline**
  - **Static Frequency**: 3.3GHz and 4.0GHz
  - **Adrenaline** [HPCA 2015]
  - **Rubik** [MICRO 2015]



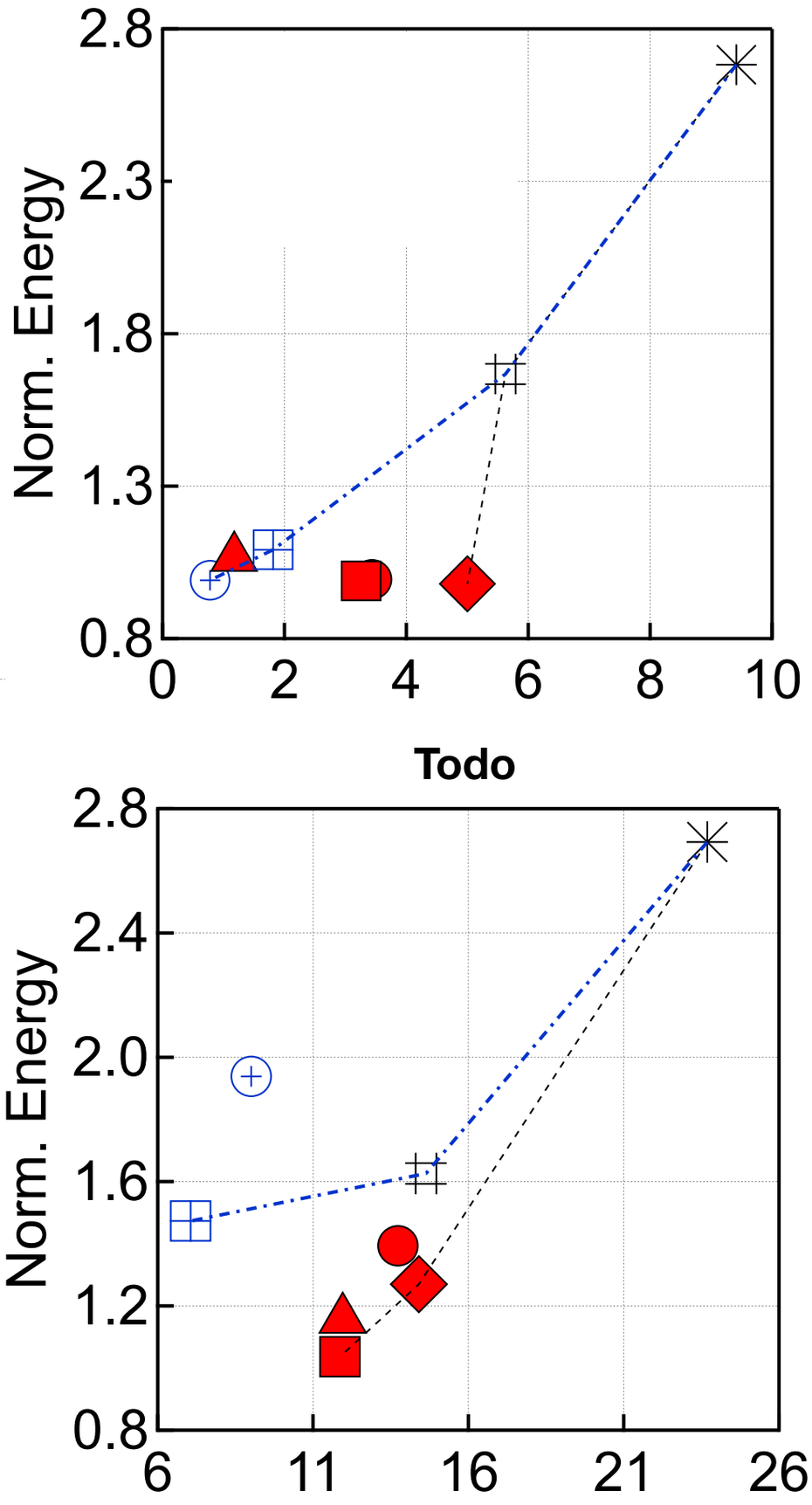
Evaluation

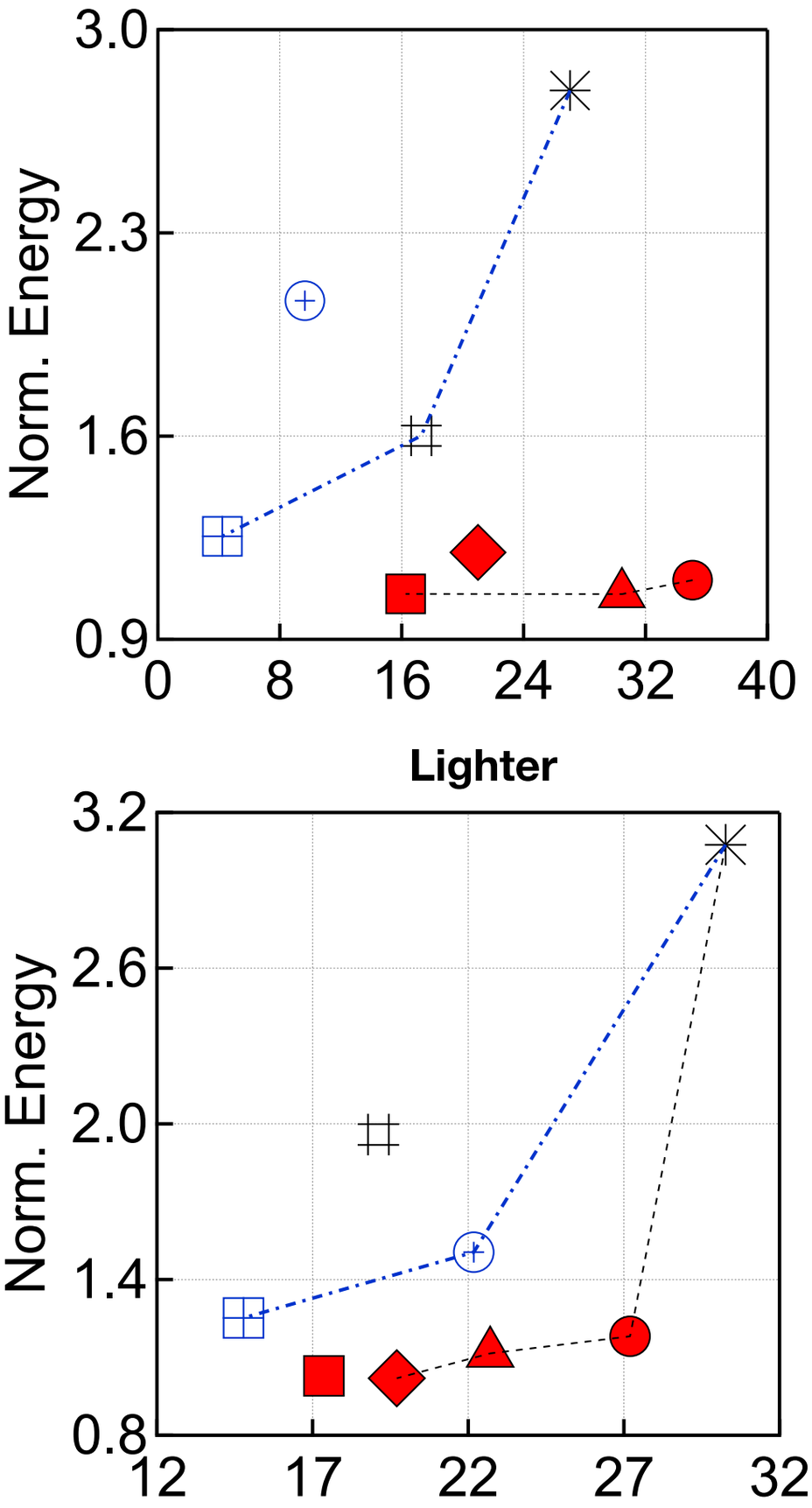
- **Our system:** normal operating frequency at 2.6 GHz, boosts to max 4.0 GHz during GC boost and Queue boost
- **Different Variants:**
  - GC Boost
  - GC Boost with GC Parameter Tuning
  - Queue Boost
  - GC Tuning + GC Boost + Queue Boost
- **Baseline**
  - **Static Frequency**: 3.3GHz and 4.0GHz
  - **Adrenaline** [HPCA 2015]
  - **Rubik** [MICRO 2015]



Combined
GC Boost
GC Boost+Tuning
QBoost

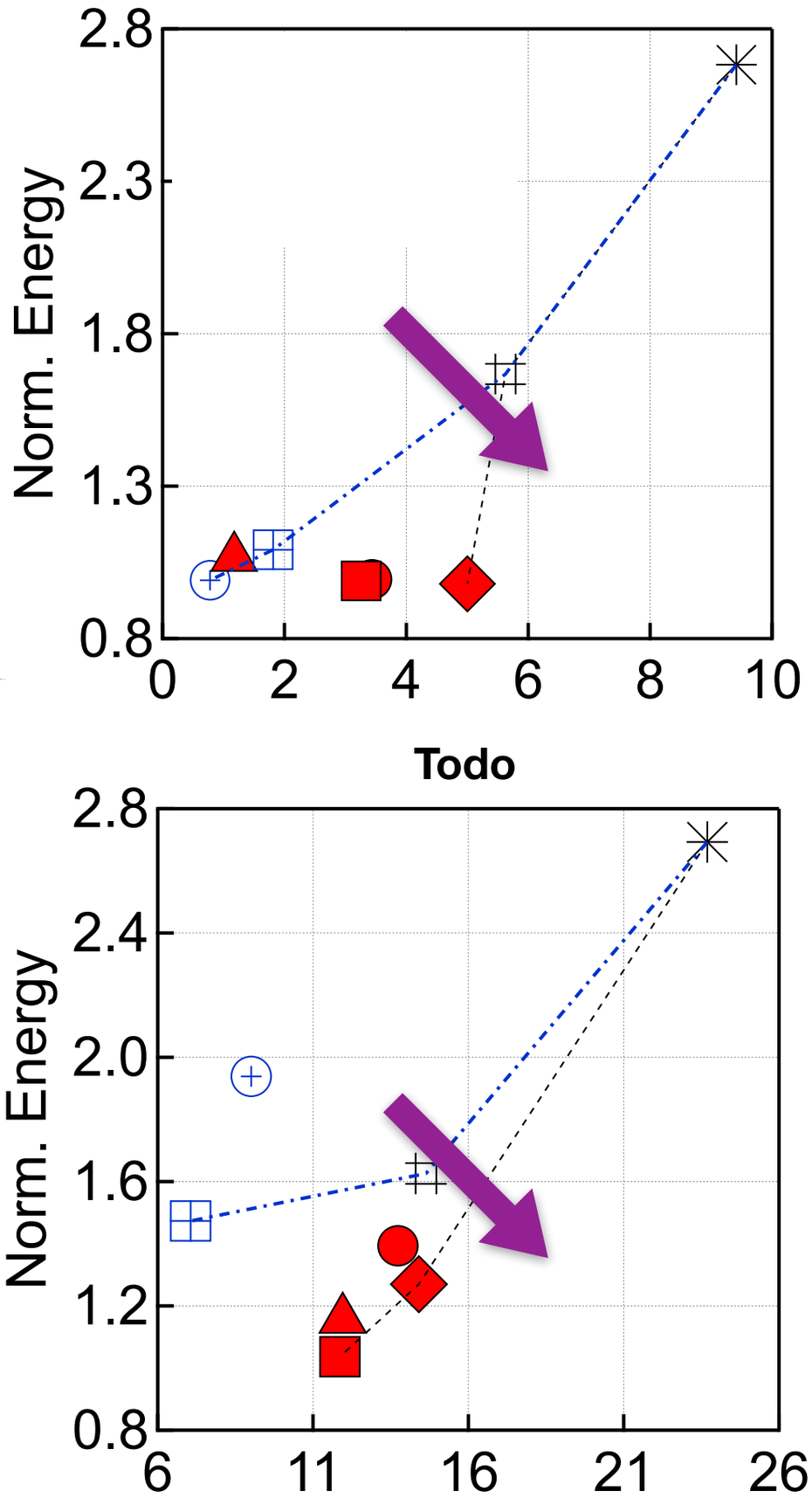
Adrenaline
Rubik
3.3 GHz
4.0 GHz

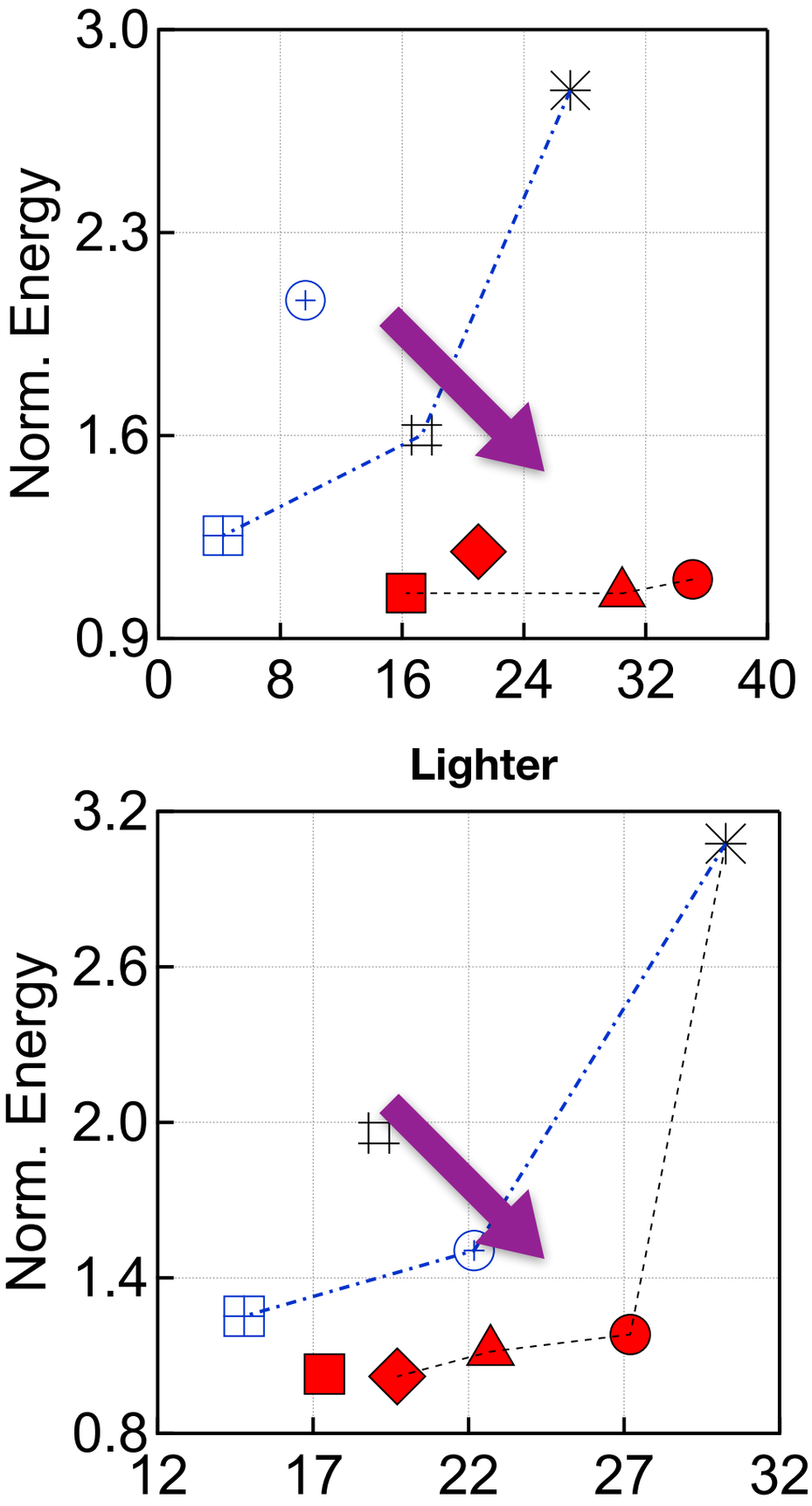


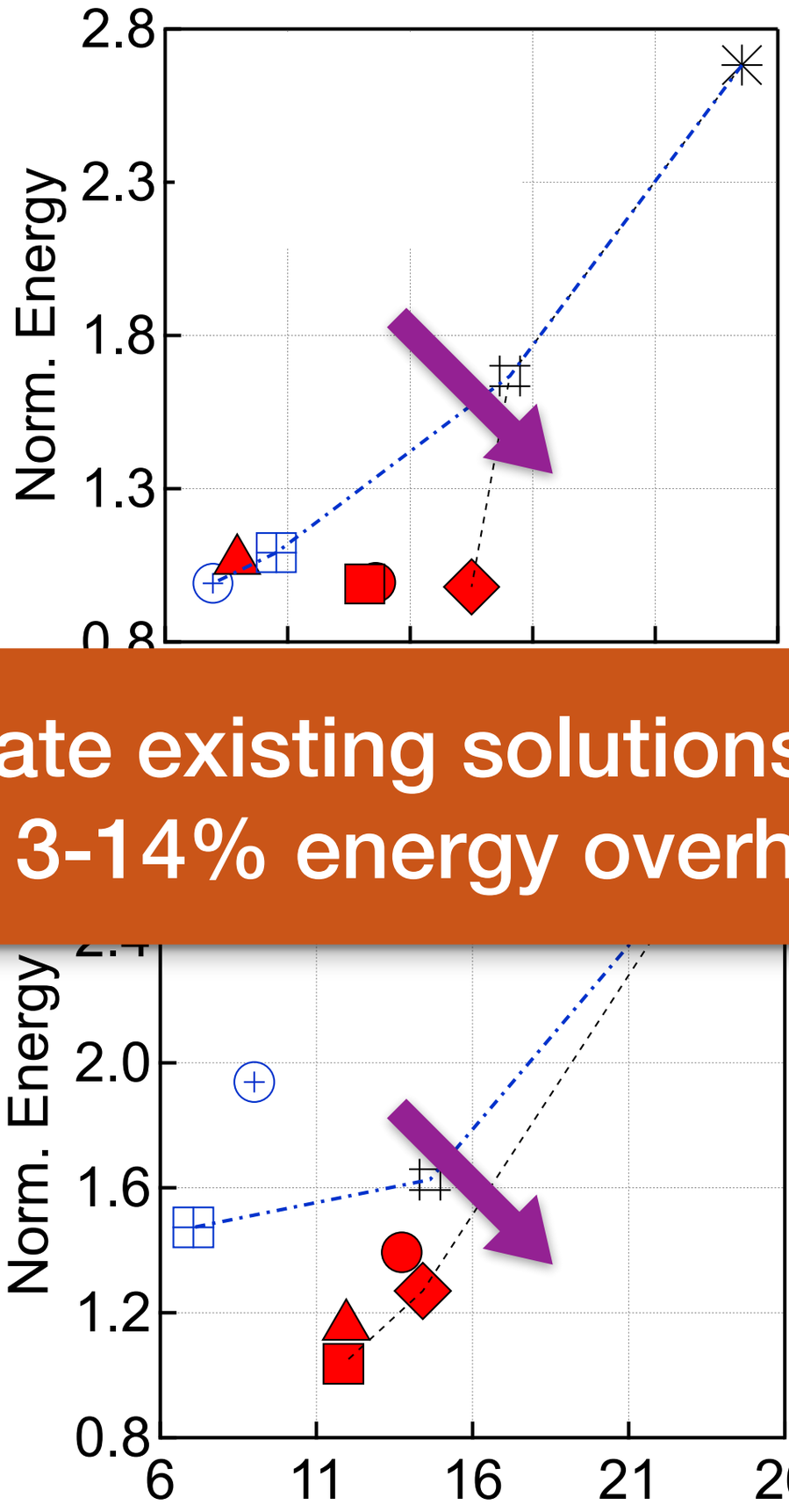


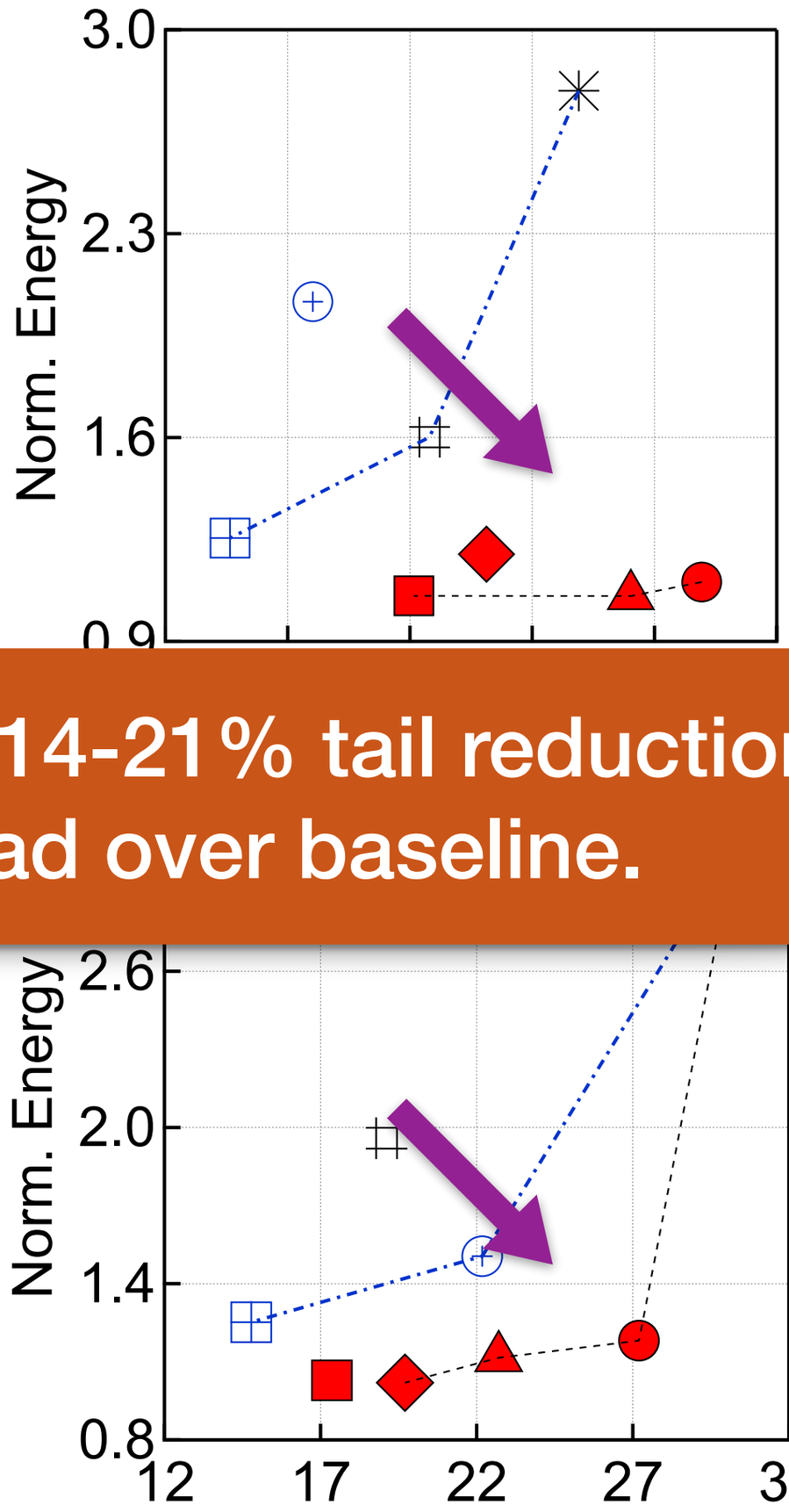
Combined
GC Boost
GC Boost+Tuning
QBoost

Adrenaline
Rubik
3.3 GHz
4.0 GHz



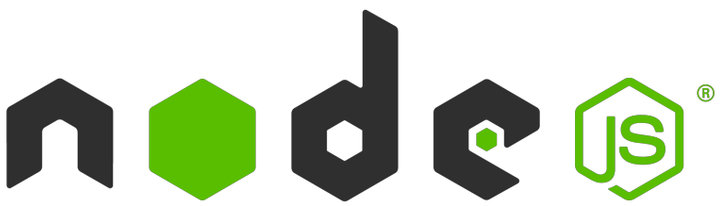






Pareto-dominate existing solutions; 14-21% tail reduction with only 3-14% energy overhead over baseline.

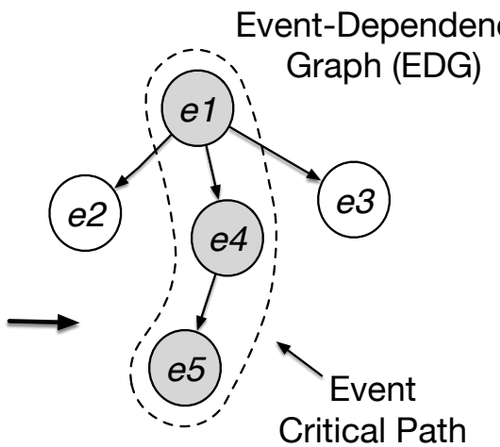
Conclusions



Node.js uniquely combines event-driven programming model and managed language runtime, presenting new landscape and challenges to tail latency optimizations.

Conclusions

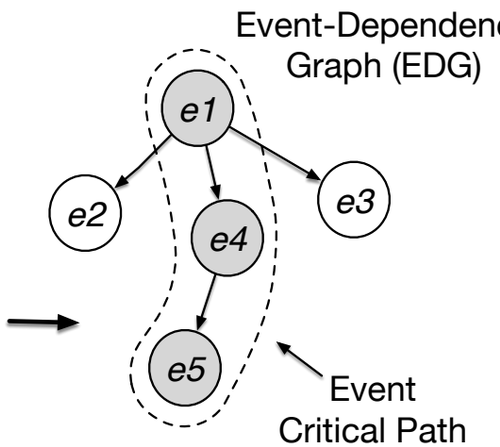
Node.js uniquely combines event-driven programming model and managed language runtime, presenting new landscape and challenges to tail latency optimizations.



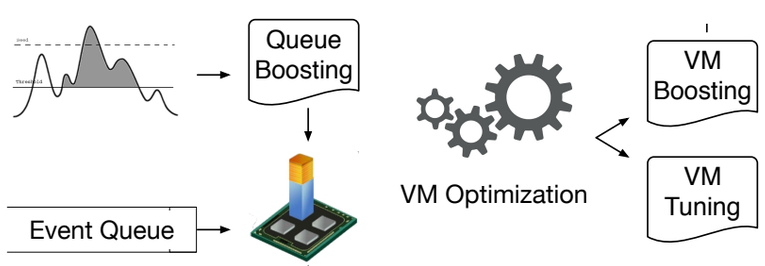
Event-dependency graph (EDG) and event-critical path (ECP) critical to deconstruct tail latency in Node.js.

Conclusions

Node.js uniquely combines event-driven programming model and managed language runtime, presenting new landscape and challenges to tail latency optimizations.



Event-dependency graph (EDG) and event-critical path (ECP) critical to deconstruct tail latency in Node.js.



Intelligently leverage existing hardware features, turbo boosting in particular, to reduce latency with little to none energy overhead.