Automation or augmentation? A few thoughts on infrastructural support for software monitoring

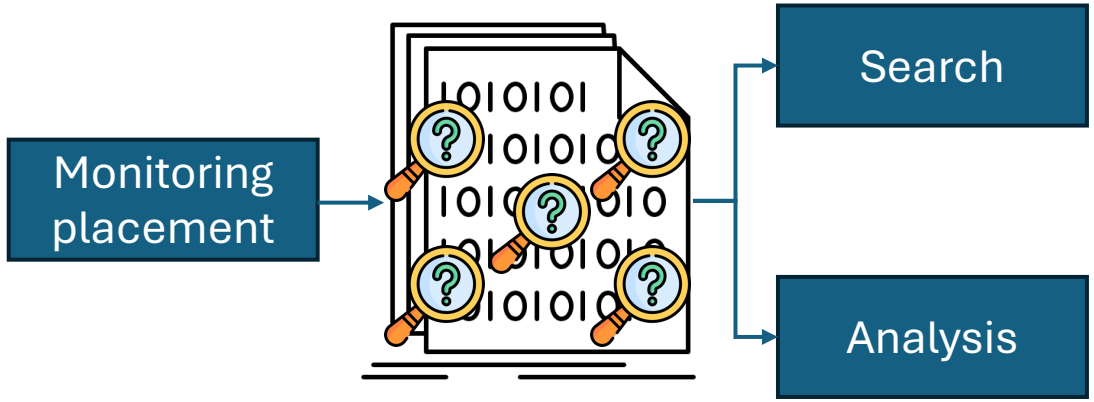
Heng Li

Dorsal Progress Report Meeting

May 2025

What is software monitoring?

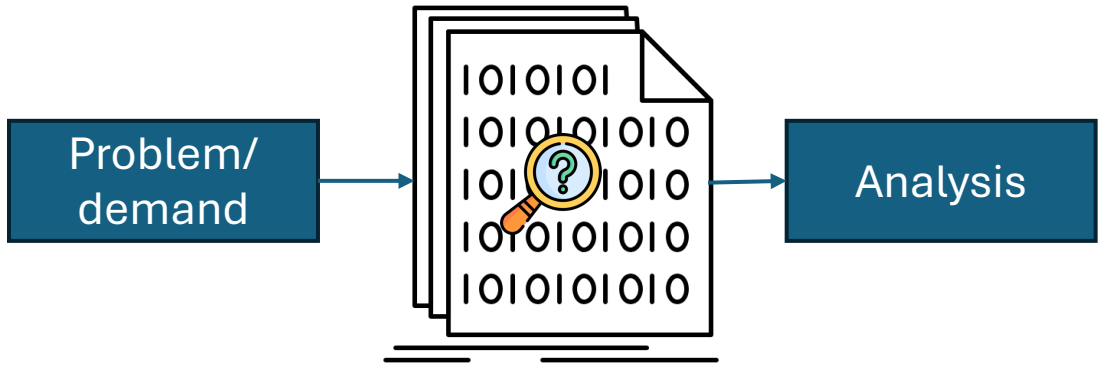
Systematic/widespread monitoring



Pros: proactive

Cons: perf./storage/analysis overhead

On-demand monitoring



Pros: lower overhead

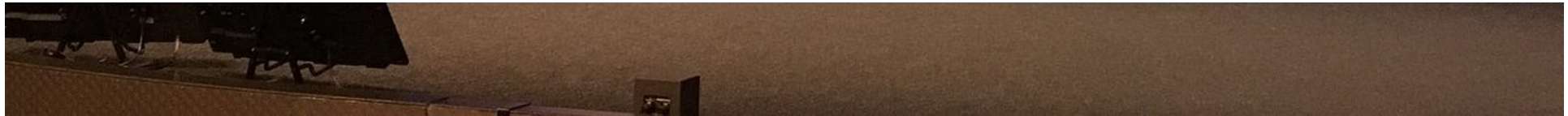
Cons: reactive


ICSE
GOTHENBURG
27 MAY – 3 JUNE
2018
icse2018.org
Big ideas
behind the Whyline
Andy J. Ko, Ph.D. Associate Professor
UNIVERSITY of WASHINGTON
PLSE
ICSE 2018
Gothenburg, Sweden

Augmentation vs. Automation

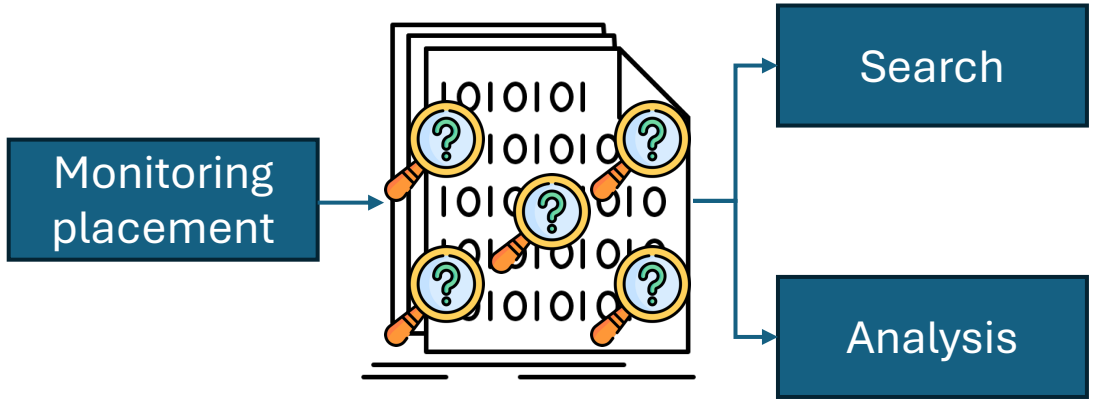
1. **Automation alone is insufficient**. Humans are always in the loop, and must always be accounted for in the design of tools, systems, and processes.

2. Augmentation > automation. Developers know far more than we think. We should be designing for augmentation, not automation, combining the insights that both developers and machines have to make much more powerful systems that automation can achieve alone.



Infrastructural support

Systematic/widespread monitoring



Pros: proactive

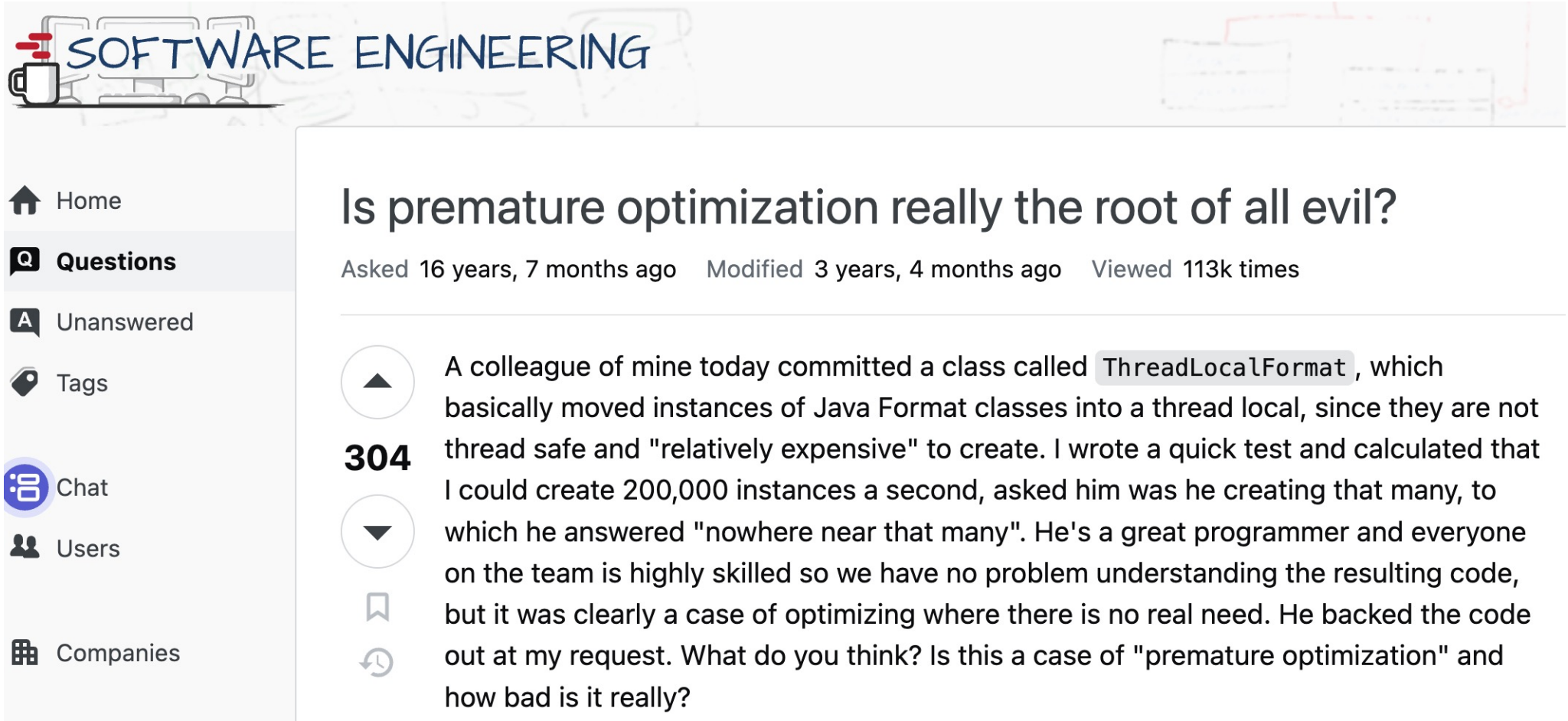
Cons: perf./storage/analysis overhead

- **Reducing overhead**
  - Low overhead tracing (LTTng)
  - Tracing optimization / adaptive tracing
  - CTF2 / log compression
- **Data preprocessing**
  - Log/trace parsing
- **Data visualization**
  - Flame Graph
  - Time Curve
  - Data Map
- **Intelligent/semantic search**
  - Vector search

Bottom-up vs. top-down approaches

Premature optimization is the root of all evil

– Donald Knuth (Turing Award Winer, 1974)



Bottom-up vs. top-down approaches

Premature optimization is the root of all evil

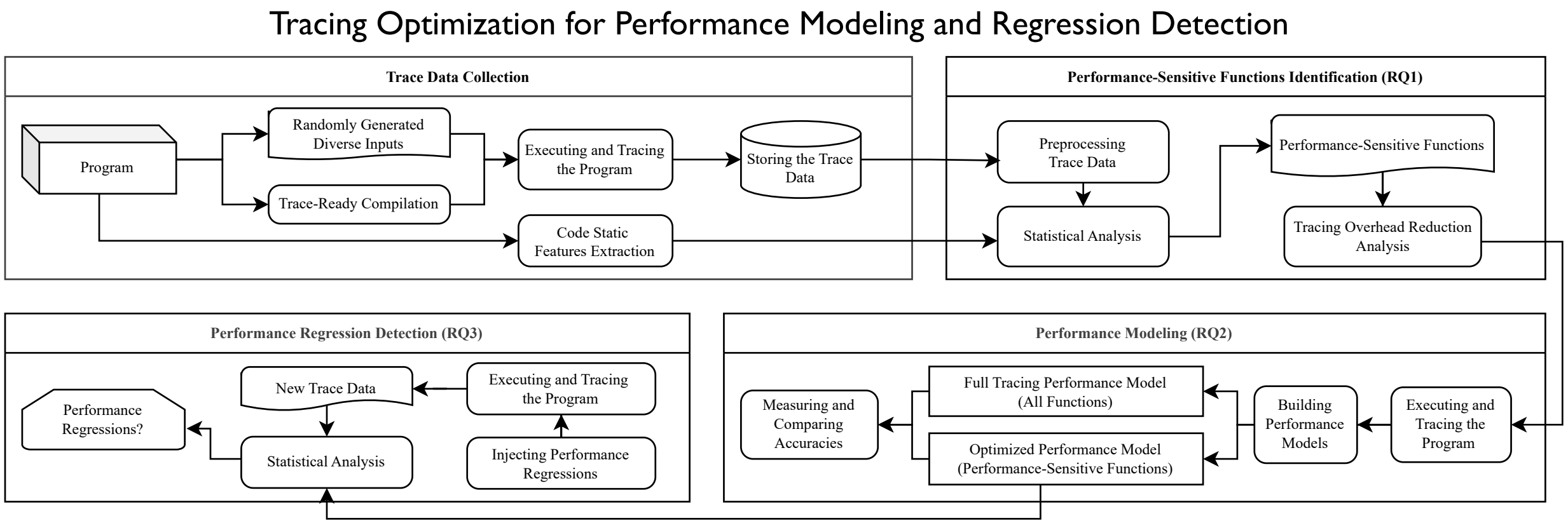
– Donald Knuth (Turing Award Winer, 1974)

Knuth, D. E. (1974). Structured programming with go to statements. *ACM Computing Surveys (CSUR)*, *6*(4), 261-301.

considered. We *should* forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.

Yet we should not pass up our opportunities in that critical 3%. A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only *after* that code has been identified. It is often a mistake to

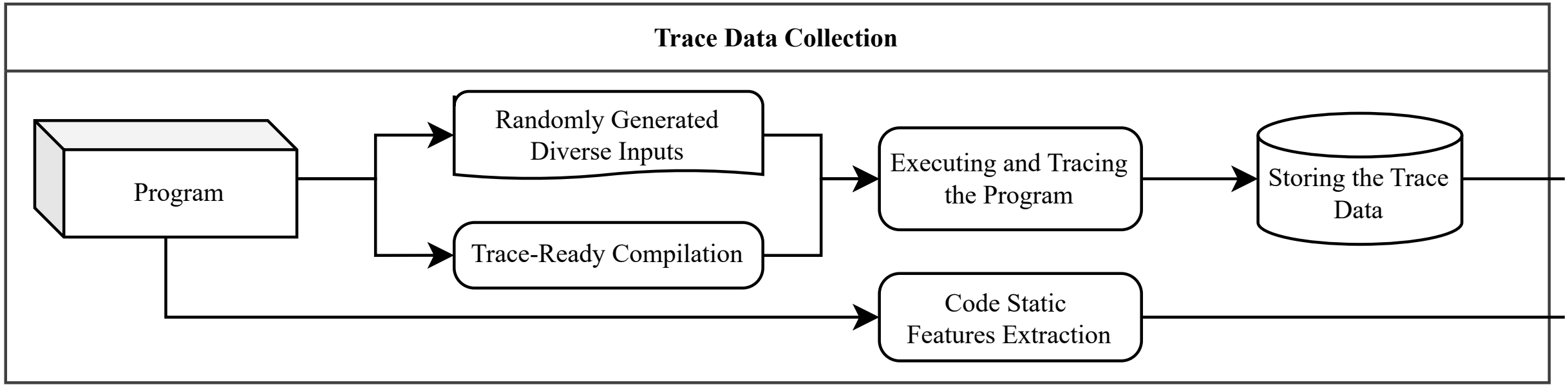
Bottom-up vs. top-down approaches



Shahedi, Kaveh, et al. "Tracing Optimization for Performance Modeling and Regression Detection." arXiv preprint arXiv:2411.17548 (2024).

Bottom-up vs. top-down approaches

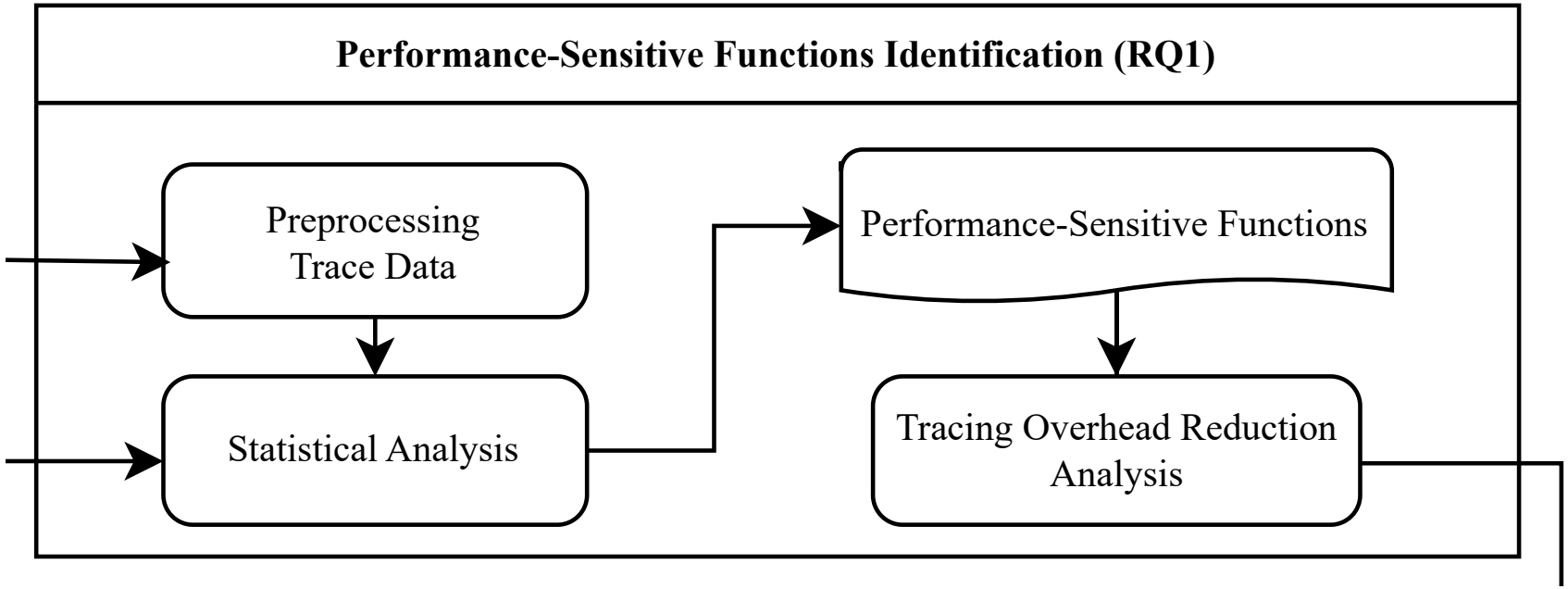
Tracing Optimization for Performance Modeling and Regression Detection



Shahedi, Kaveh, et al. "Tracing Optimization for Performance Modeling and Regression Detection." arXiv preprint arXiv:2411.17548 (2024).

Bottom-up vs. top-down approaches

Tracing Optimization for Performance Modeling and Regression Detection

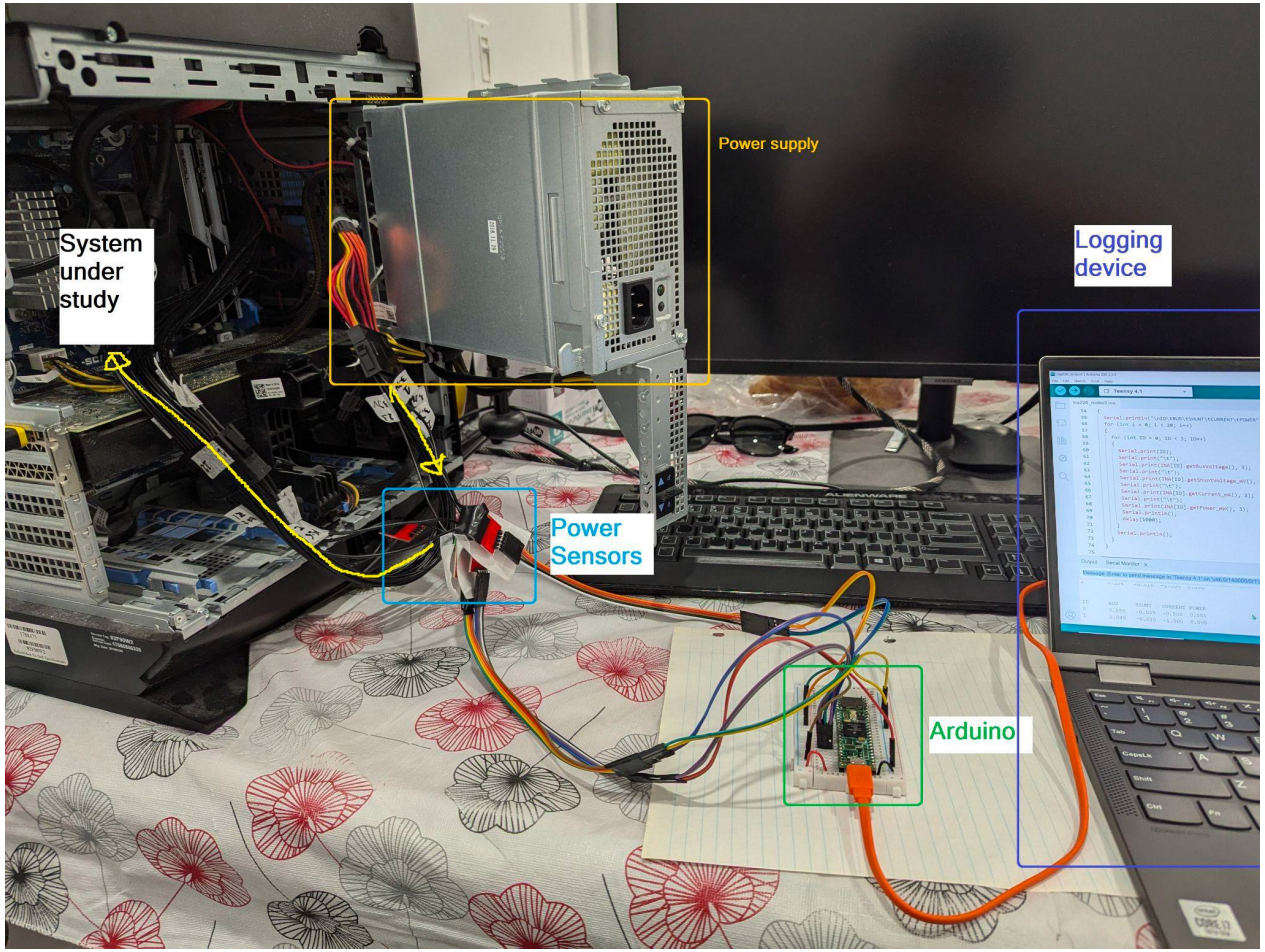


Top-down approach to identify

Only a small portion of functions (as low as1.0%) statistically significantly contribute to system performance variations

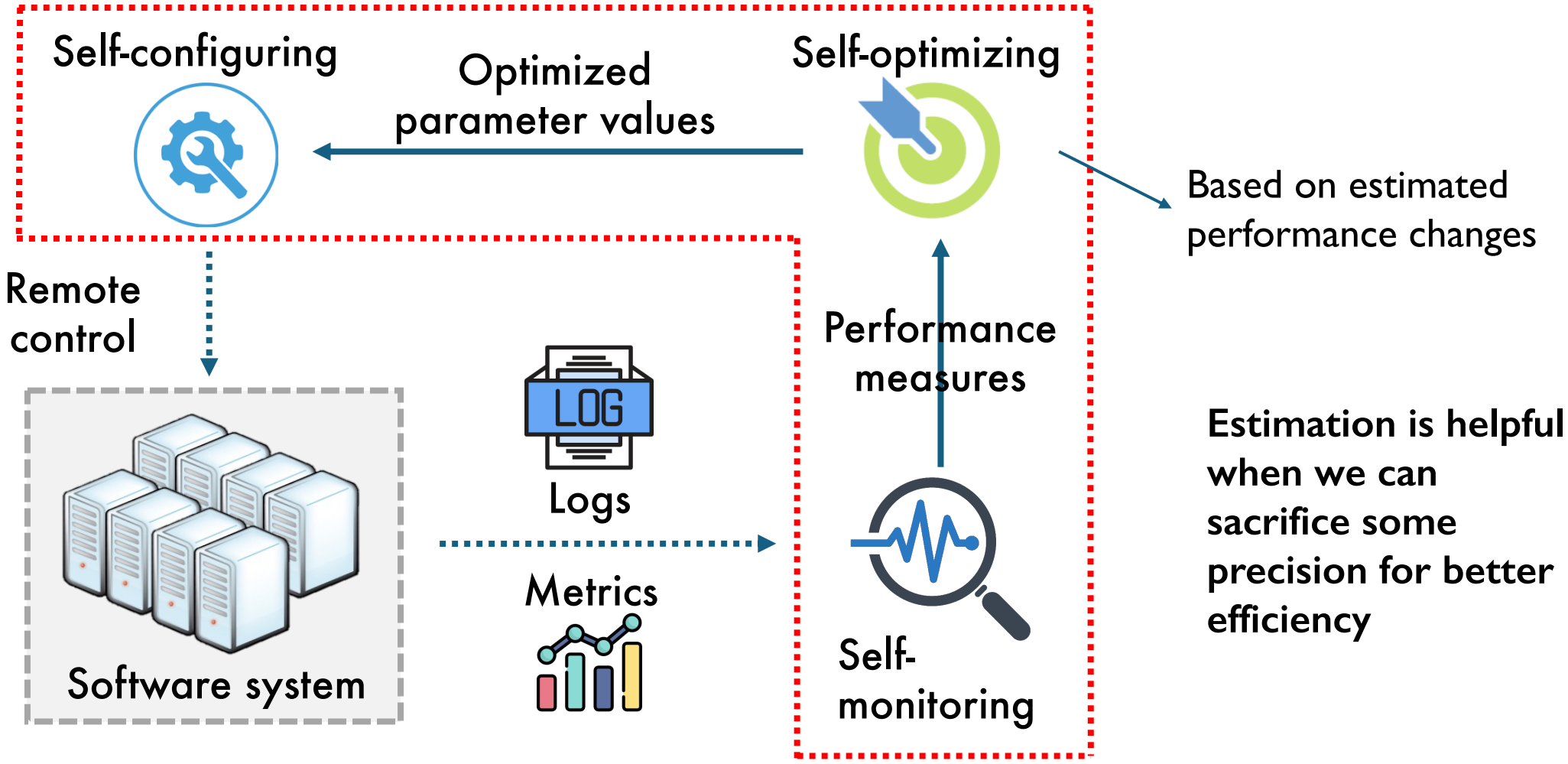
Shahedi, Kaveh, et al. "Tracing Optimization for Performance Modeling and Regression Detection." arXiv preprint arXiv:2411.17548 (2024).

Measurement vs. estimation



Andy's measurement setup

Measurement vs. estimation

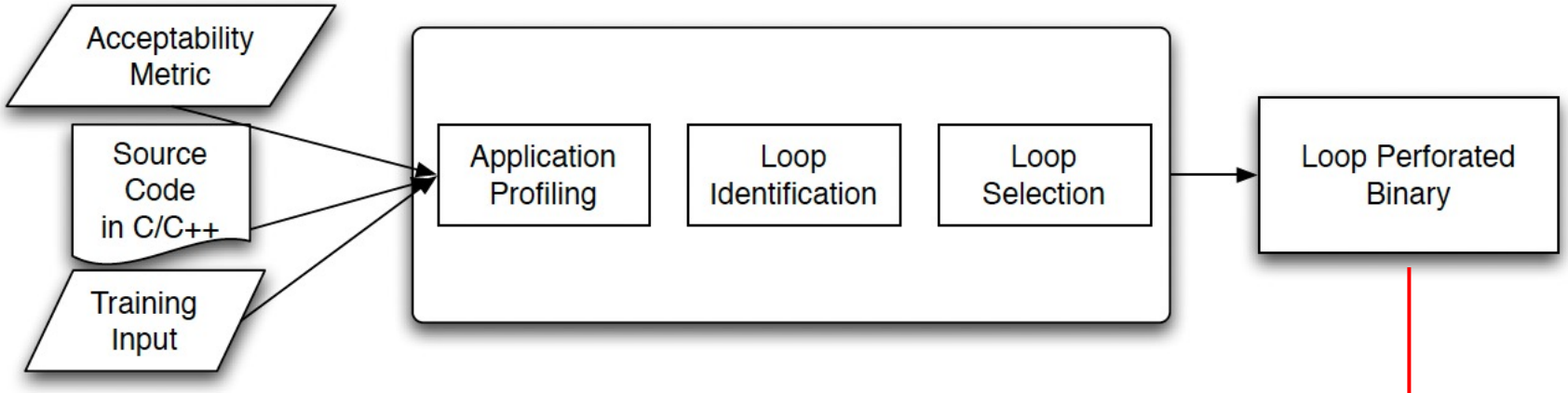


Estimation is helpful when we can sacrifice some precision for better efficiency

Li, Heng, et al. "Adopting autonomic computing capabilities in existing large-scale systems: An industrial experience report." ICSE-SEIP, 2018.

Approximate computing: software is continuous?

Can we leverage the continuity of software for efficient monitoring?




```
for( i = 0; i < h; i+=4 ) { /* ... */ }
```

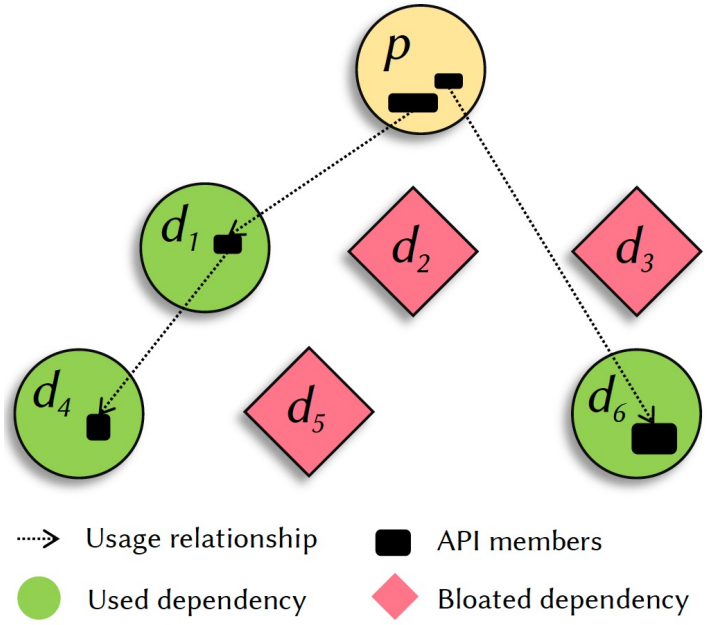
to

```
for( i = 0; i < h; i+=4 ) {
   if (doPerforate(i, environment)) continue;
   //...
}
```

Hoffmann, Henry, et al. "Using code perforation to improve performance, reduce energy consumption, and respond to failures." (2009).

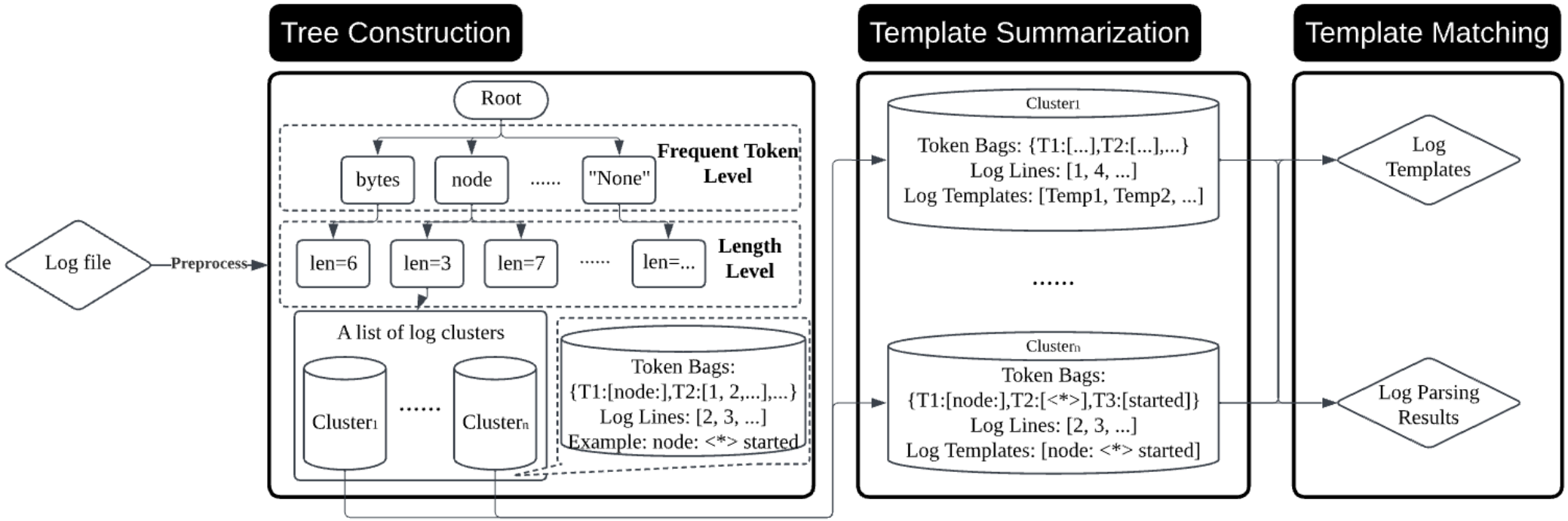
Complex/expensive/untrusted dependences may cause issues

- Software supply chain issues
  - Maintainability
  - Security
  - Bloated dependencies
- Dependency on external services such as LLM servers
  - Monetary costs
  - Stability/availability/evolution



Soto-Valero, César, Thomas Durieux, and Benoit Baudry. "A longitudinal analysis of bloated java dependencies." ESEC/FSE 2021.

LLM vs. classical: a classical log parser performing better than LLM-based approaches



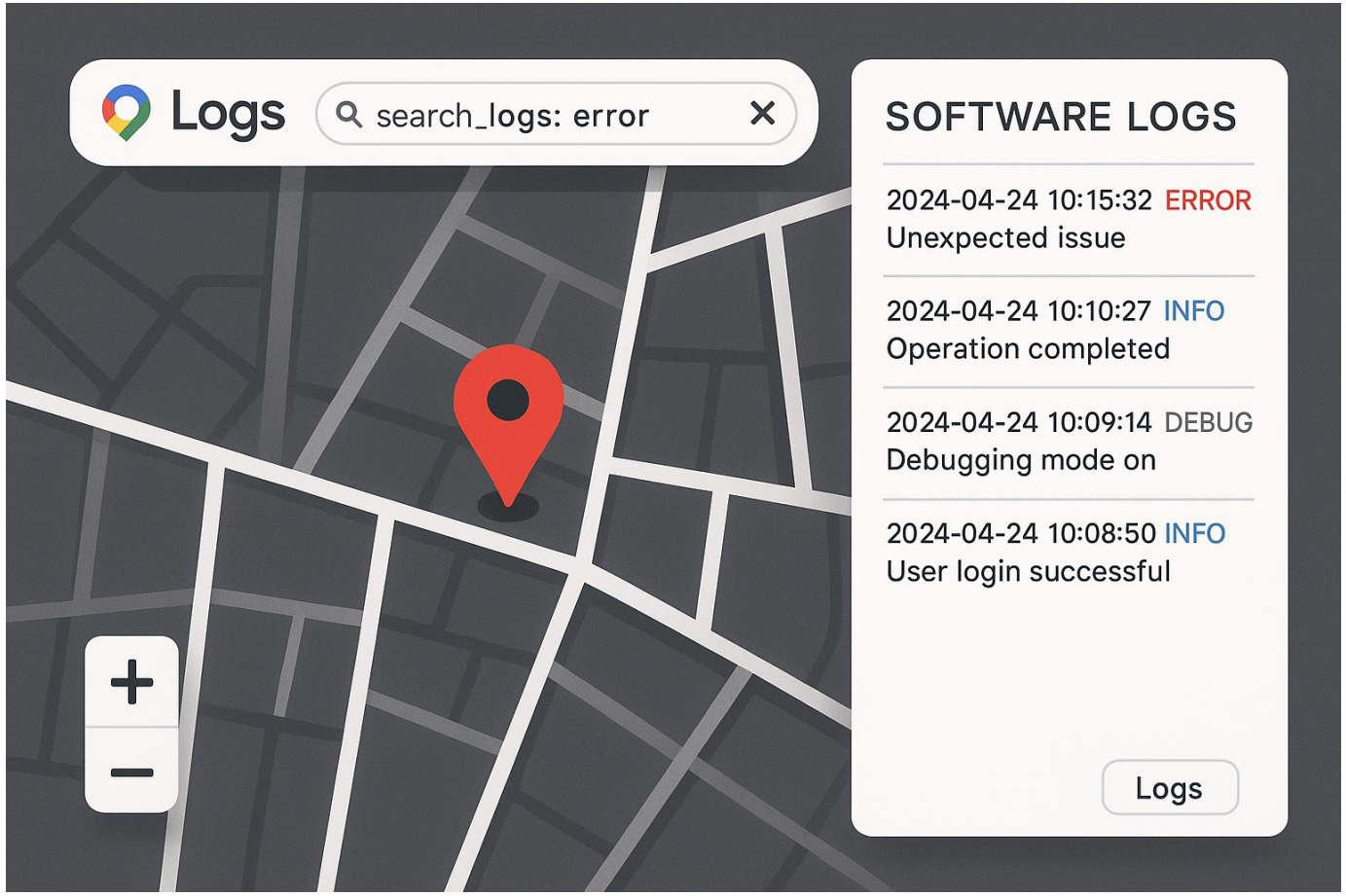
Qiaolin Qin et al. Plug it and Play on Logs: A Configuration-free Log Parser. Under review.

Augmenting developers' capacity through visualization

- What tool/app helps you the most in your everyday life?

Augmenting developers' capacity through visualization

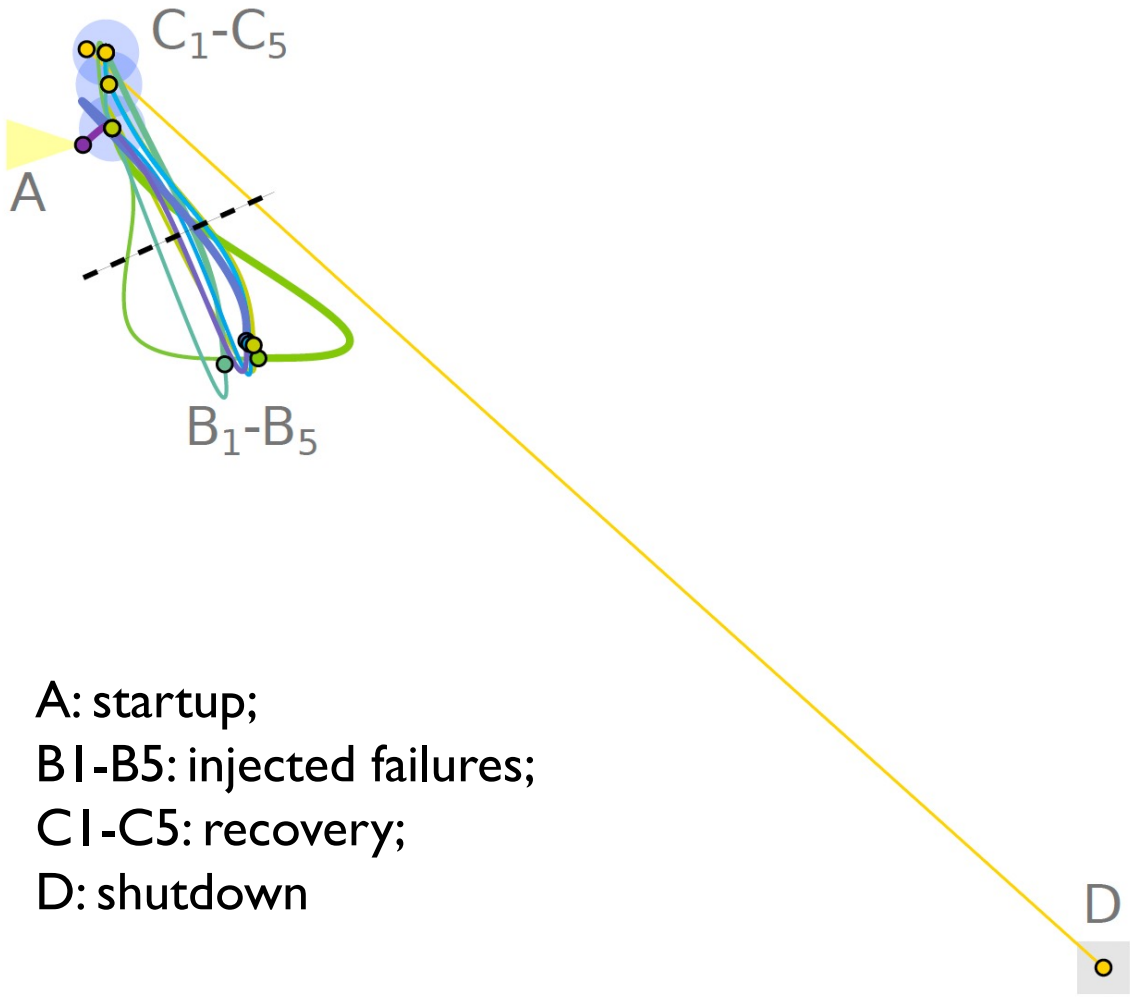
- Imagine we have a "Google Map" for log/trace data (e.g., log map)



(Figure by ChatGPT)

Augmenting developers' capacity

- Time curve visualization for log data (Spark logs)



A: startup;
B1-B5: injected failures;
C1-C5: recovery;
D: shutdown

Dmytro Borysenkov et al. Analyzing Logs of Large-Scale Software Systems using Time Curves Visualization. SANER 2025

Summary

- Automation or augmentation
- Bottom-up (premature optimization?) vs. top-down approaches
- Estimation/approximation is helpful when we can sacrifice some precision for better efficiency
- Complex/expensive/untrusted dependences may cause issues
- Augmenting developers' capacity through visualization