StackMine – Performance Debugging in the Large via Mining Millions of Stack Traces

Shi Han[1], Yingnong Dang[1], Song Ge[1], Dongmei Zhang[1], and Tao Xie[2]

Software Analytics Group, Microsoft Research Asia[1] North Carolina State University[2]

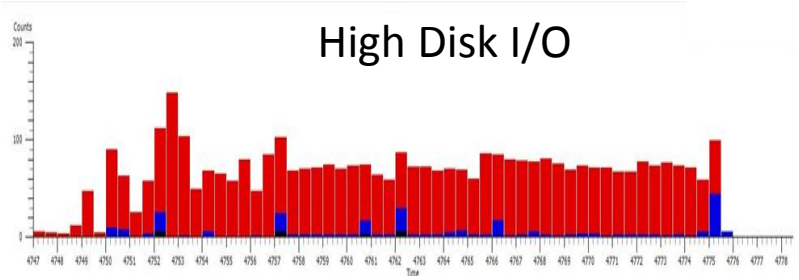
June 6th, 2012

Microsoft Research
微软亚洲研究院

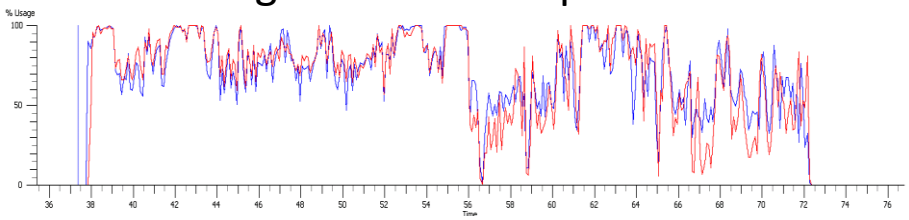
Performance Issues in the Real World

- One of top user complaints
- Impacting large number of users every day
- High impact on usability and productivity

High Disk I/O

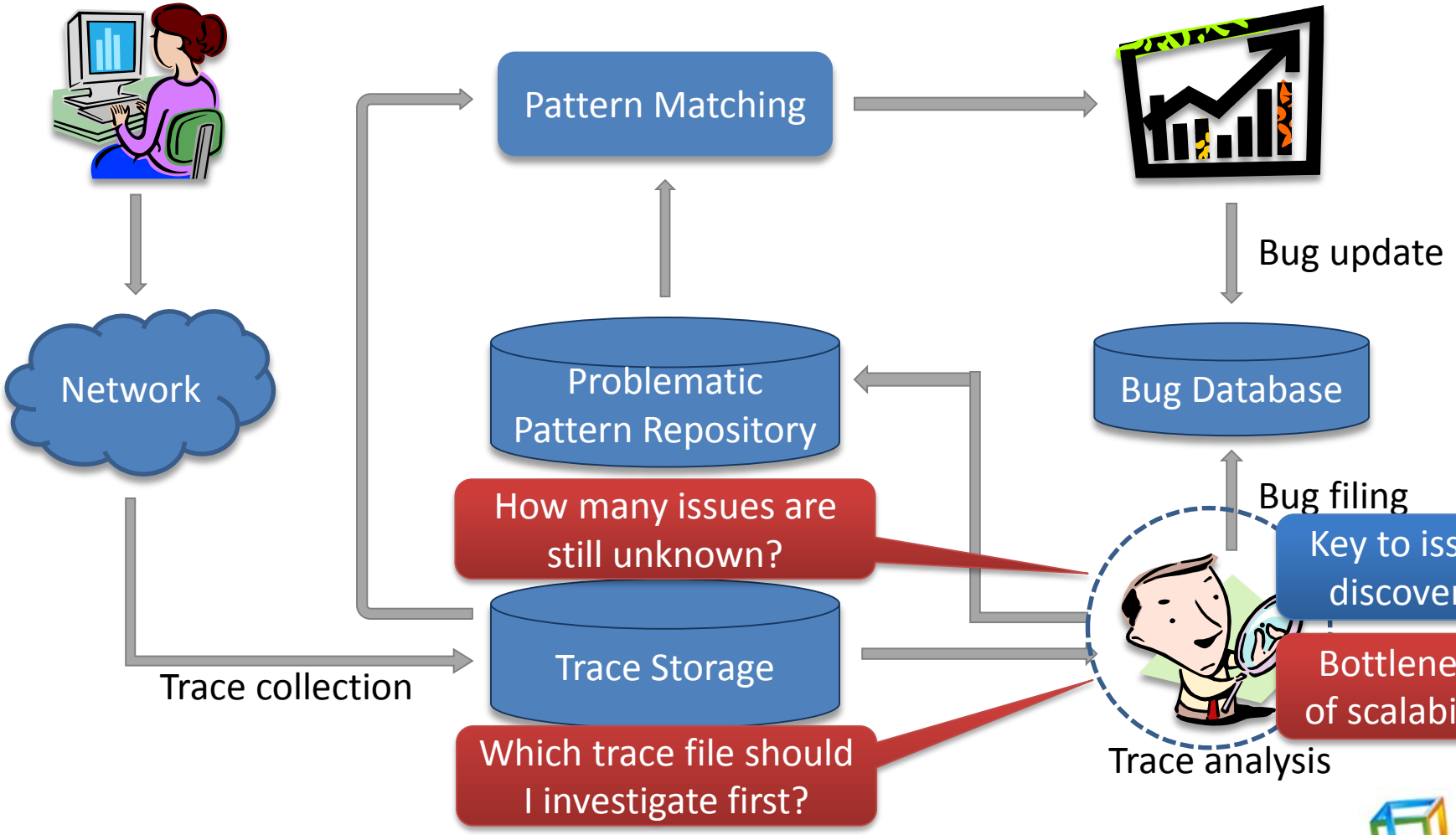


High CPU consumption



Given **limited** time and resource **before** software release, **development-site** testing and debugging become **insufficient** to ensure satisfactory software performance.

Performance Debugging in the Large



Problem Definition

Input: Runtime traces

collected from millions of users

Output: Program execution patterns causing the most impactful performance problems

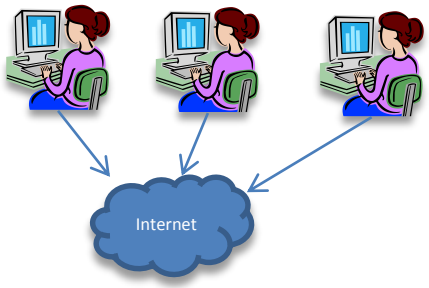
inspected by performance analysts

Goal

Conduct **systematic discovery & analysis of program execution patterns**

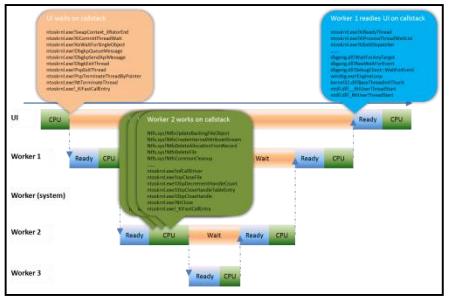
- Efficient handling of large-scale trace sets
- Automatic discovery of new patterns
- Effective prioritization of investigation

Challenges



Large-scale trace data

- TBs of trace files and increasing
- Millions of events in single trace stream



Highly complex analysis

- Numerous program runtime combinations triggering performance problems
- Multi-layer runtime components from application to kernel being intertwined

Combination of expertise

- Generic machine learning tools without domain knowledge guidance do not work well

Wait callstack

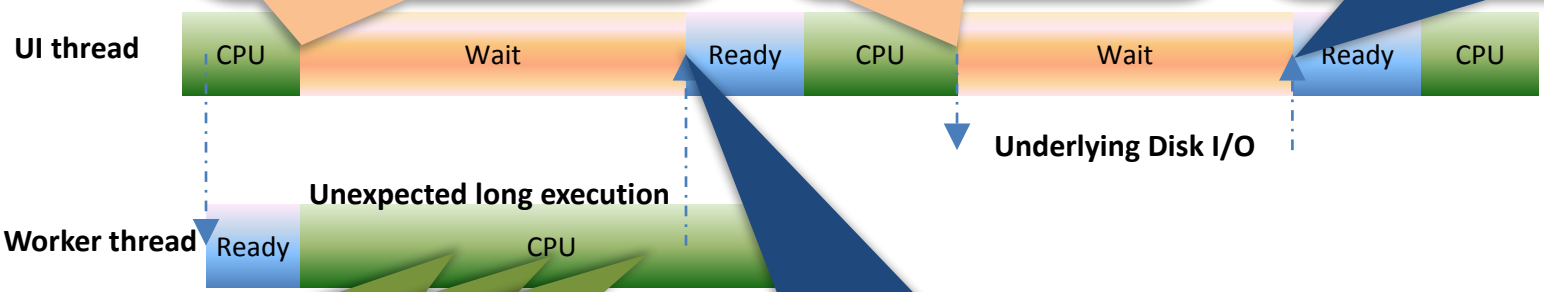
ntdll!UserThreadStart
Browser! Main
…
Browser!OnBrowserCreatedAsyncCallback
…
BrowserUtil!ProxyMaster::GetOrCreateSlave
BrowserUtil!ProxyMaster::ConnectToObject
…
rpc!ProxySendReceive
…
wow64!RunCpuSimulation
wow64cpu!WaitForMultipleObjects32
wow64cpu!CpupSyscallStub
…

Wait callstack

ntdll!UserThreadStart
Browser! Main
…
ntdll!LdrLoadDll
…
nt!AccessFault
nt!PageFault
…

ReadyThread callstack

nt!KiRetireDpcList
nt!ExecuteAllDpcs
…
nt!IopfCompleteRequest
…
nt!SetEvent
…

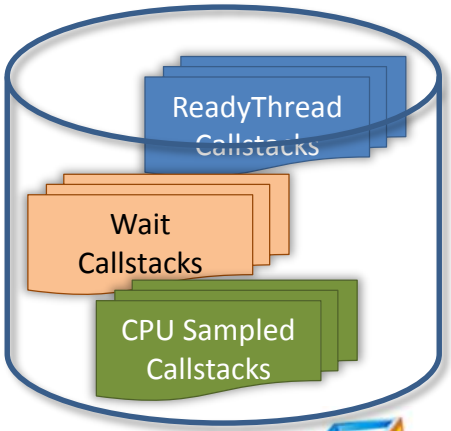


CPU sampled callstack

ntdll!UserThreadStart
…
ntdll!WorkerThread
…
ole!CoCreateInstance
…
ole!OutSerializer::UnmarshalAtIndex
ole!CoUnmarshalInterface
…

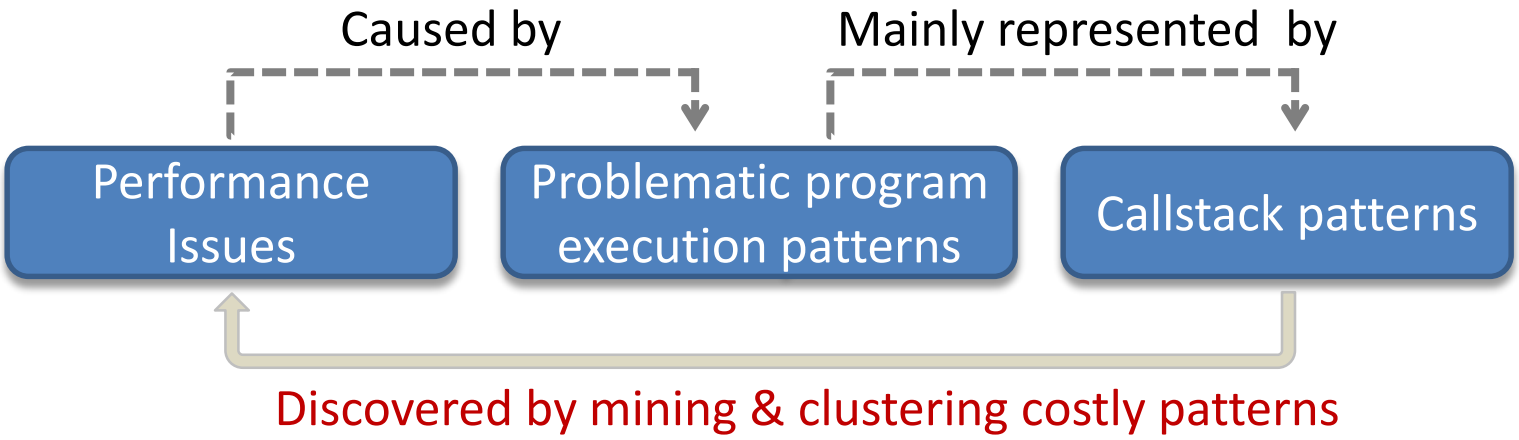
ReadyThread callstack

ntdll!UserThreadStart
…
rpc!LrpcIoComplete
…
user32!PostMessage
…
win32k!SetWakeBit
nt!SetEvent
…

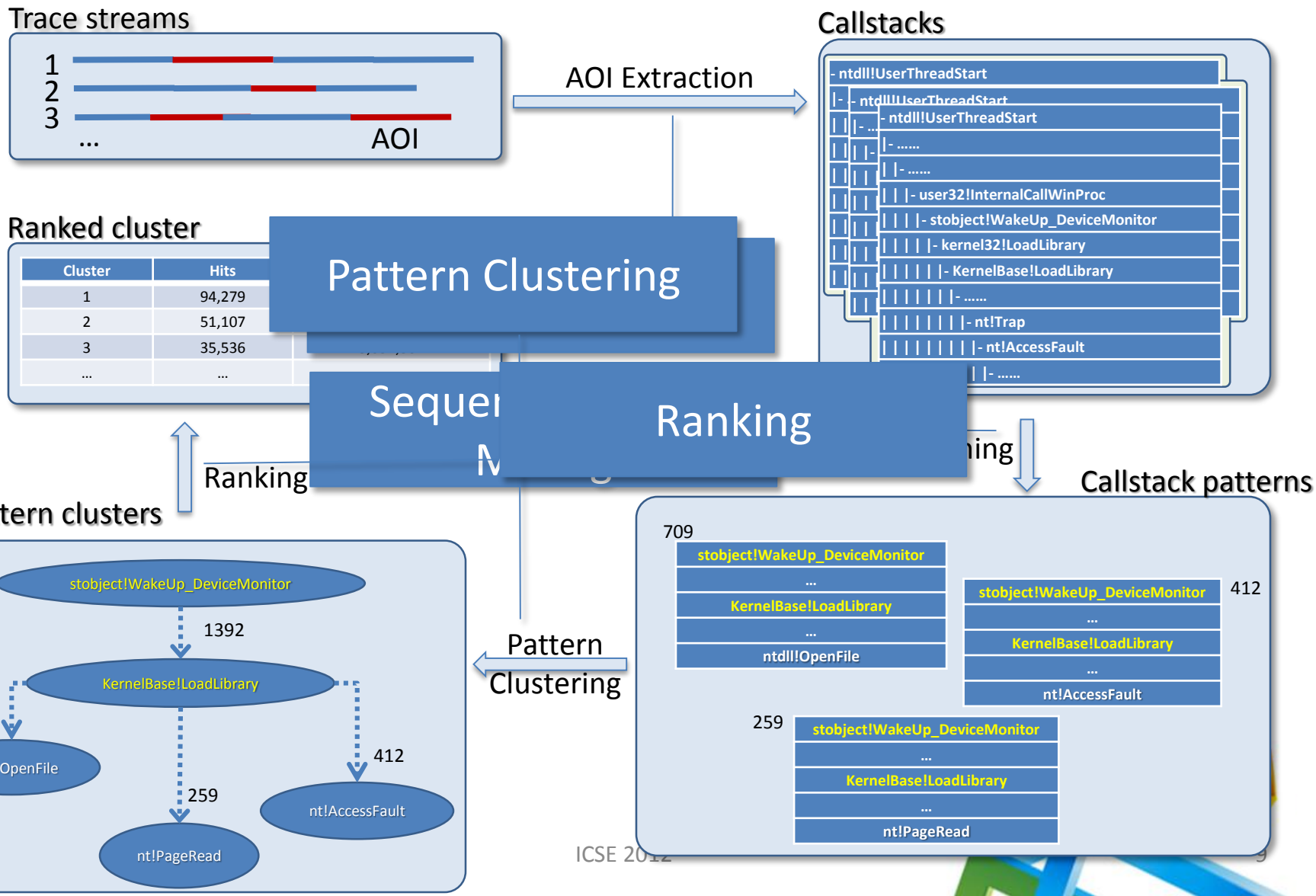


Approach

Formulated as a callstack mining and clustering problem

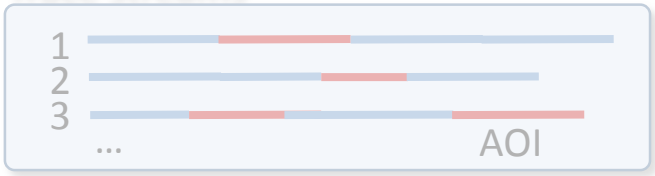


Approach – Workflow



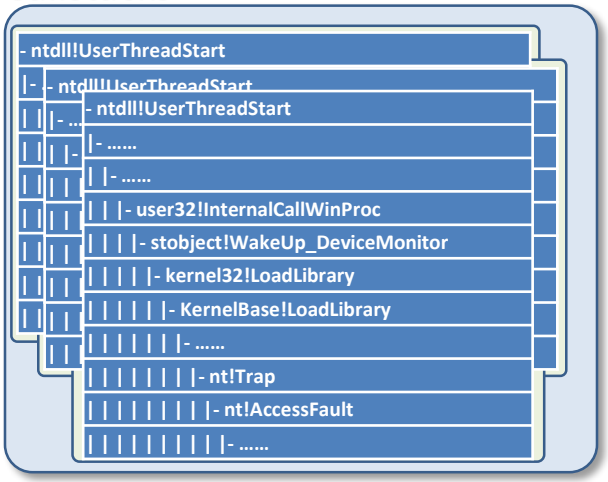
Approach – AOI Extraction

Trace streams



AOI Extraction

Callstacks

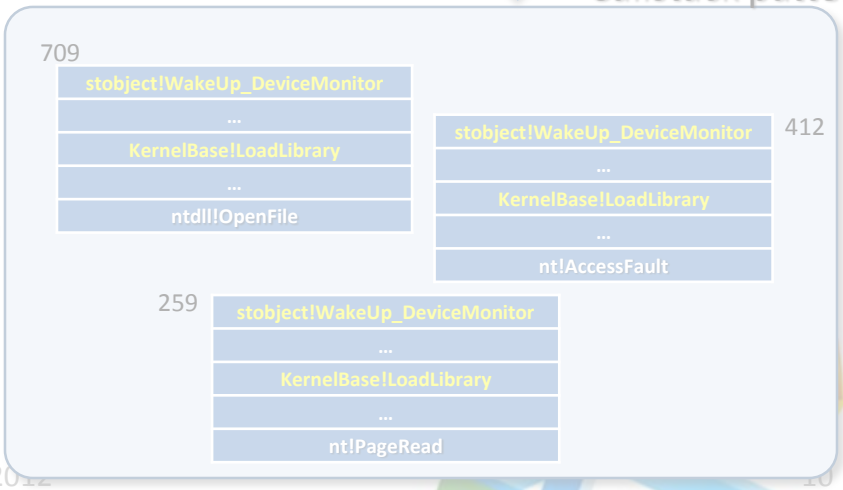


Ranked cluster

Cluster	Hits	Total Wait time (ms)
1	94,279	927,824
2	51,107	561,416
3	35,536	3,051,307
...	...	

Sequence Pattern Mining

Callstack patterns



Pattern Clustering

Ranking

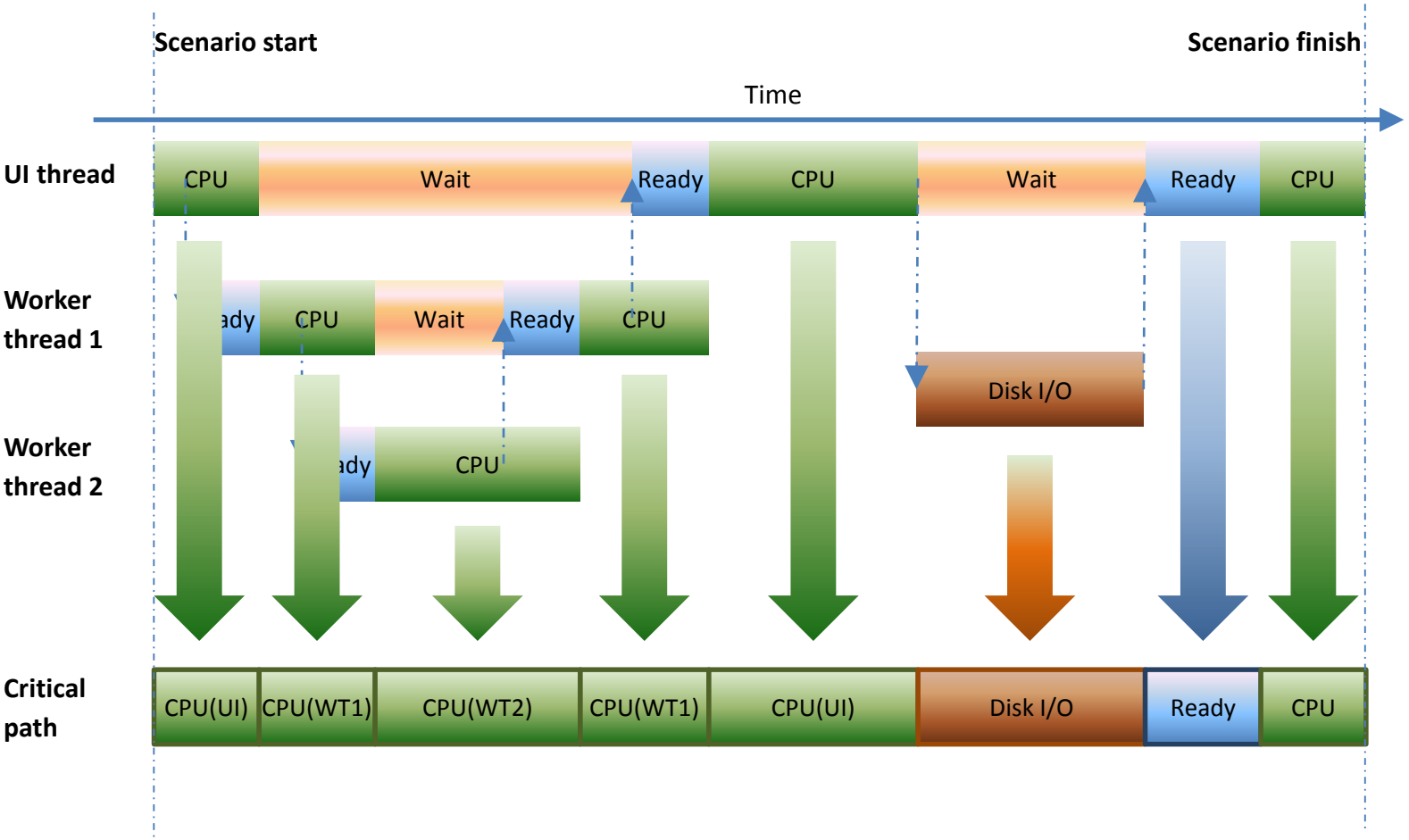
Pattern clusters

Approach – AOI Extraction Motivation

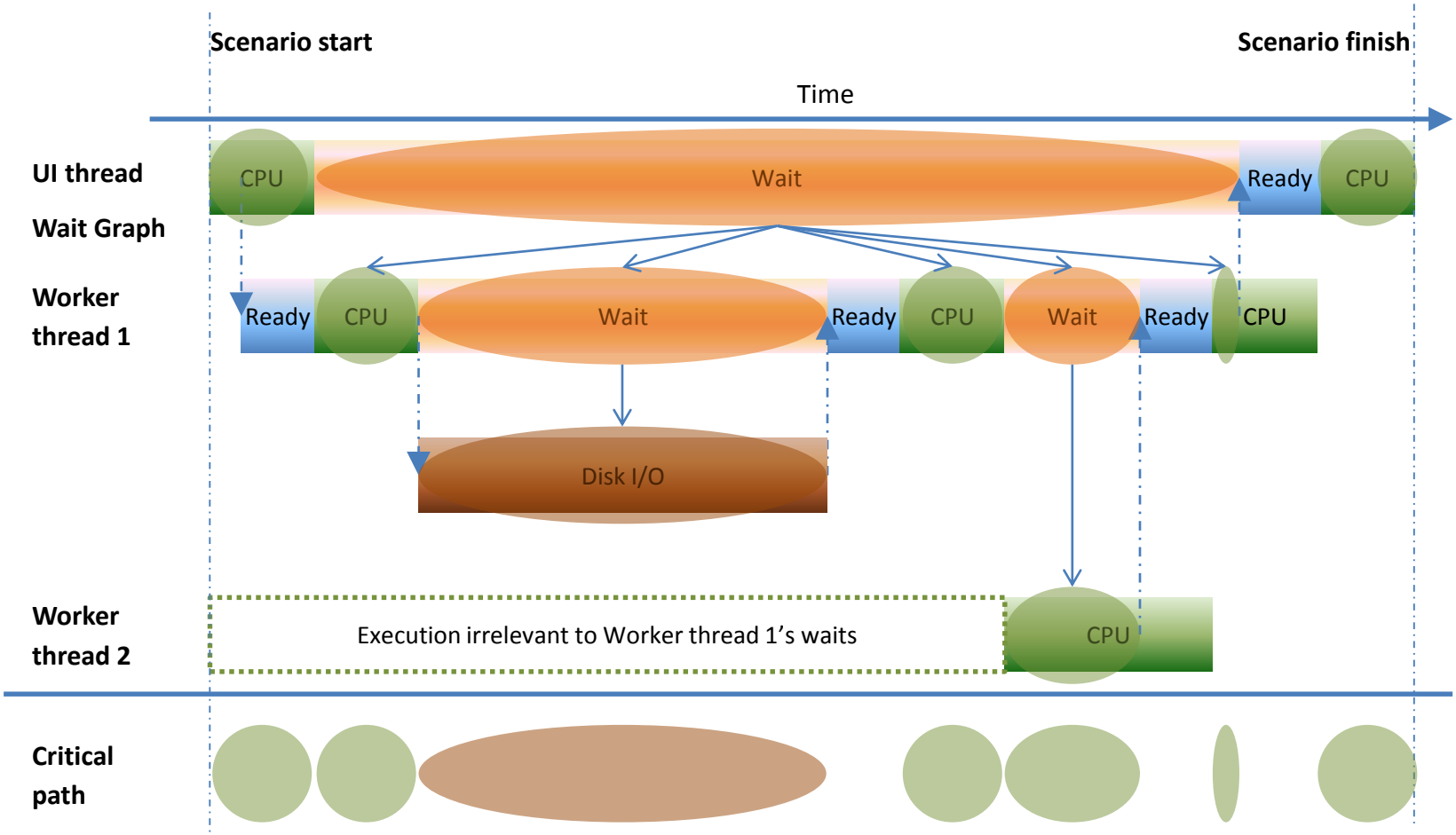
Runtime traces capture both

- Relevant executions for performance issue
  - E.g., executions relevant to browser-tab creation
- Irrelevant executions for performance issues
  - E.g., executions of concurrently executed IM
  - Noisy data for mining
  - Huge investigation scope induced

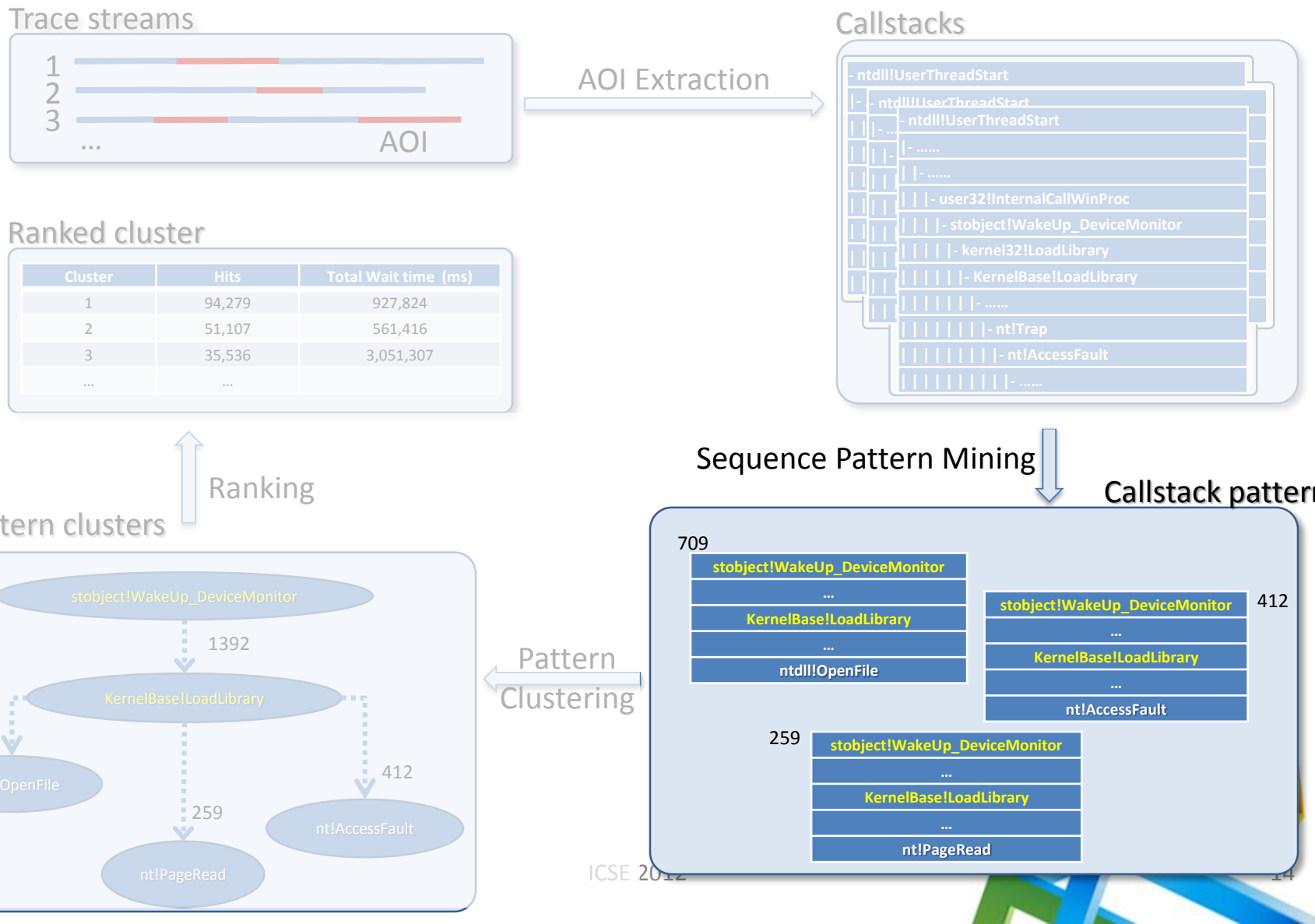
Approach – AOI Extraction Critical Path



Approach – AOI Extraction Wait Graph



Approach – Callstack Pattern Mining



Approach – Callstack Pattern Mining
Costly Sequence Pattern

- Example

Sequence	Wait time
A B	4 ms
A B C D F	1 ms
A B C E F	3 ms
A B G F	2 ms

Cost threshold T
5 ms (or 50% of total)

Costly sequence pattern miner

All Patterns	Support
A, B, AB	10
F, AF, BF, ABF	6

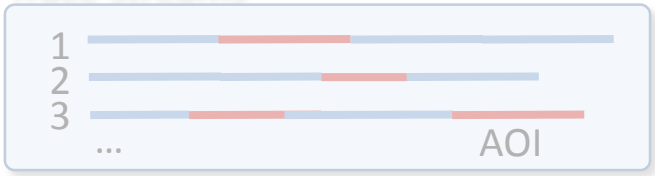
Closed Patterns	Support
~~A, B,~~ AB	10
~~F, AF, BF,~~ ABF	6

Maximal Patterns	Support
~~A, B, AB~~	~~10~~
~~F, AF, BF,~~ ABF	6

- Non-consecutive costly sub-sequence as callstack pattern
- Costly maximal patterns are compact

Approach – Callstack Pattern Clustering

Trace streams



AOI Extraction

Callstacks

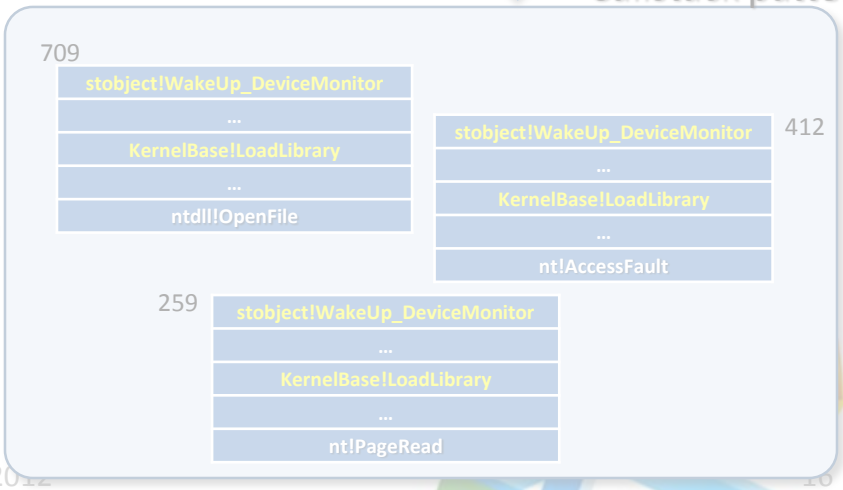
Ranked cluster

Cluster	Hits	Total Wait time (ms)
1	94,279	927,824
2	51,107	561,416
3	35,536	3,051,307
...	...	

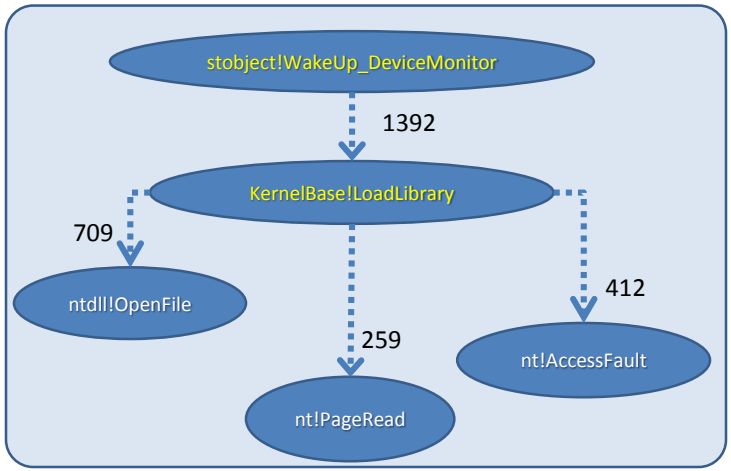
Ranking

Sequence Pattern Mining

Callstack patterns



Pattern clusters

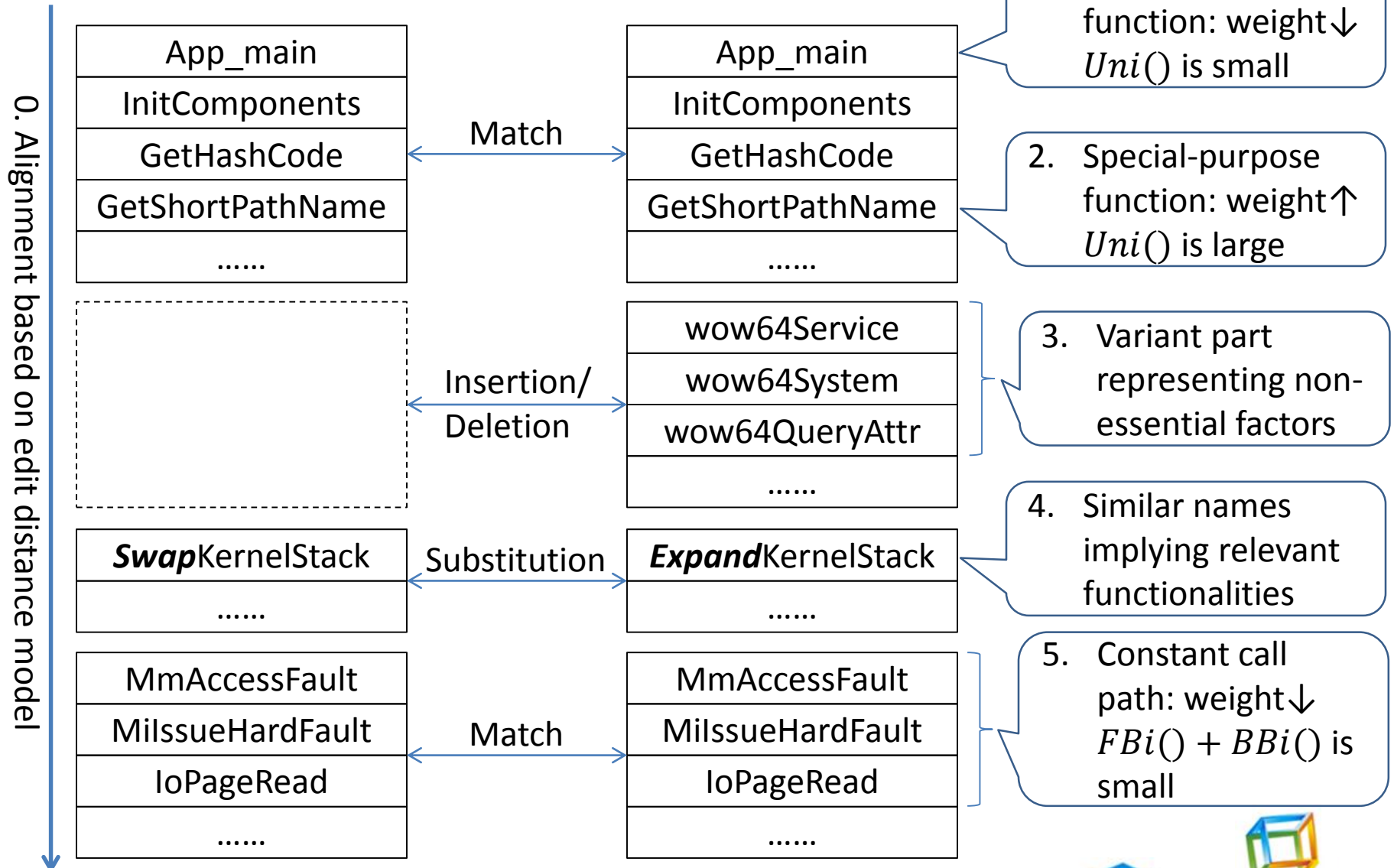


Pattern Clustering

Approach – Callstack Pattern Clustering

- Motivation
  - Same issue often reflected by variant patterns
  - Defect often hidden in invariant parts of variant patterns
- Goal
  - Precise measurement of issue impact for better prioritization
  - Comprehensive issue representation with pattern variations for quick and precise fixing

Approach – Callstack Pattern Clustering Similarity Model



Technical Highlights

- Machine learning for system domain
  - Formulate the discovery of problematic execution patterns as callstack mining & clustering
  - Systematic mechanism to incorporate domain knowledge
- Interactive performance analysis system
  - Parallel mining infrastructure based on HPC + MPI
  - Visualization aided interactive exploration

Evaluation – Windows 7 Study

- Task: since Dec 2010, a continued effort to improve Windows performance
  - **Analysts** from one performance analysis team for Microsoft Windows
  - **Hunt** for hidden performance bugs that caused common impact on Windows Explorer UI response
  - **Based** on over 6,000 trace streams
- Data
  - 921 qualified out of 1,000 randomly sampled trace streams
  - 181 million callstacks in total
    - 140 million wait callstacks
    - 41 million CPU sampled callstacks

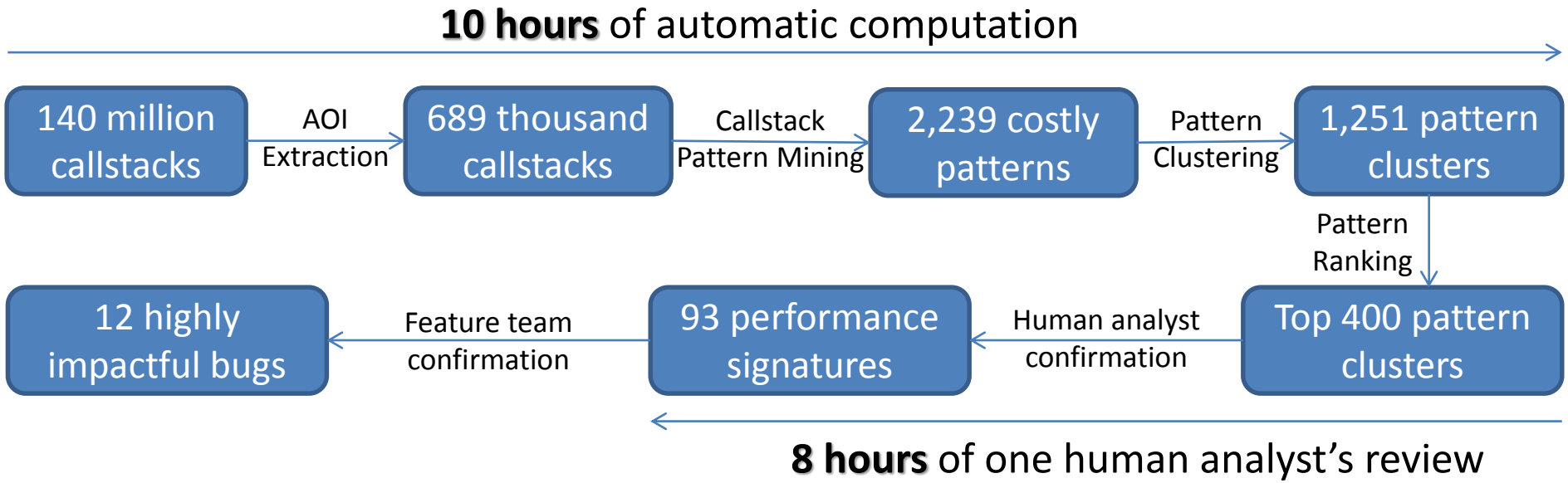
Evaluation – Windows 7 Study Research Questions

- *RQ1.* How **much** does StackMine **improve** practices of performance debugging in the large?

- *RQ2*. How **well** do the derived performance signatures **capture** performance bottlenecks?
-
- *RQ3.* How **much** does StackMine **outperform** alternative techniques?

Evaluation – Windows 7 Study
RQ1. Overall Improvement of Practices

- Traditional approach – would take 20~60 days
- Using StackMine – 18 hours



Evaluation – Windows 7 Study
RQ2. Performance Bottleneck Coverage

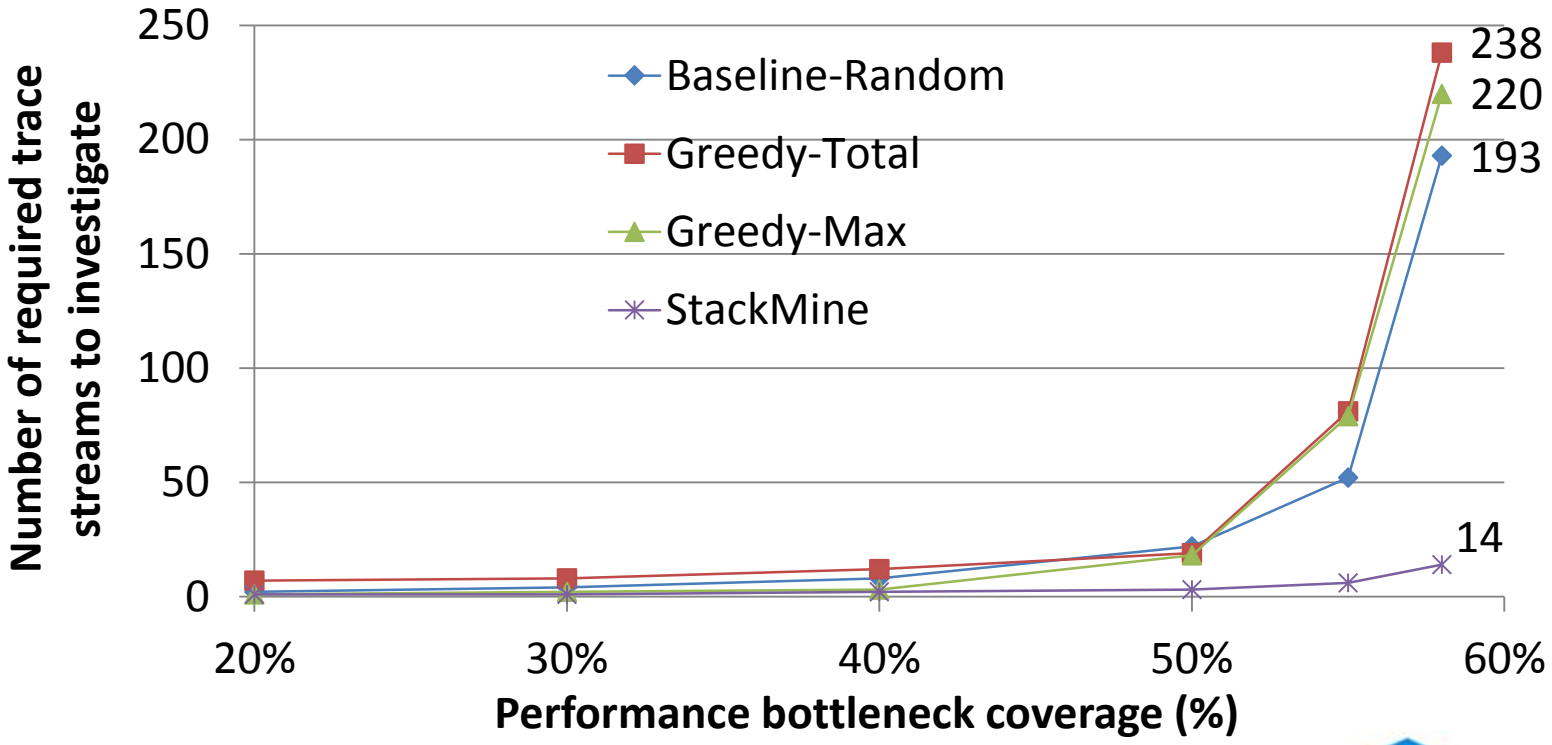
- Performance Bottleneck Coverage (PBC) of a set of performance signatures

$$PBC = \frac{Total\ time\ of\ performance\ signatures}{Total\ time\ of\ collected\ trace\ streams}$$

- The higher PBC achieved, the lower possibility that high-impact performance bugs remain not captured
- **58.26%** PBC achieved by the 93 signatures

Evaluation – Windows 7 Study
RQ3. Comparison with Alternative Techniques

StackMine requires only **7.2%, 5.8%,** and **6.3%** of trace streams required by the other three techniques



Impact

*"We believe that the MSRA tool is highly valuable and much more efficient for mass trace (100+ traces) analysis. For **1000 traces**, we believe the tool **saves us 4-6 weeks** of time to create new signatures, which is quite a significant productivity boost."*

- from Development Manager in Windows

Highly effective new issue discovery on Windows mini-hang

Continuous impact on future Windows versions

Conclusion

- The first formulation and real-world deployment of performance debugging in the large
  - as a data mining problem on callstacks
- A mining-clustering mechanism for reducing costly-pattern mining results
  - based on domain-specific characteristics of callstacks
- Industrial impact on using StackMine in performance debugging in the large for Microsoft Windows

Acknowledgment

- Our partners in Microsoft product teams

- The researchers from Microsoft Research

Q&A

Thank you!