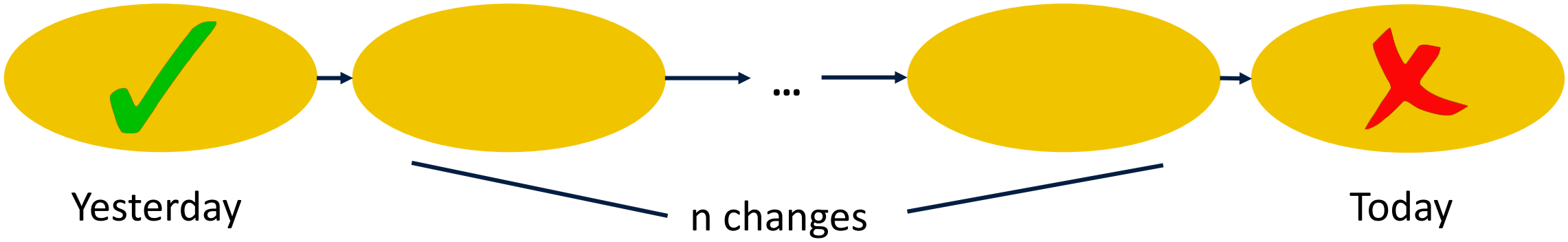
# Yesterday, my program worked. Today, it does not. Why?

# (Delta Debugging)


Andreas Zeller
Presented by Amani Alkayyali
October 29, 2018
EECS 582

# Delta Debugging

- Differences determine cause for program behavior
- Slower than conventional debugging
- Automatic debugging



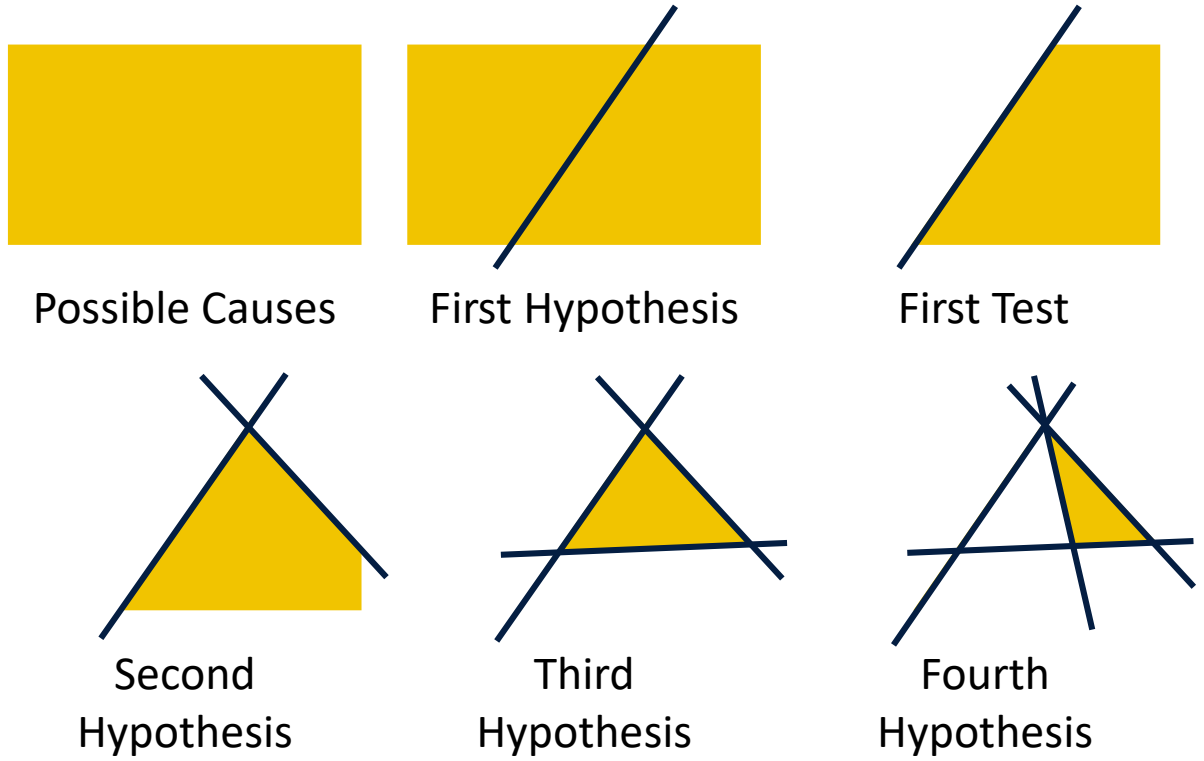
# Scientific Method


- Code
- Run
- Problem Report
- More Runs

- Hypothesis **Supported** - Refine
- Hypothesis **Rejected** - Create New

# Delta Debugging

- Technique, not a tool
- Similar to unit testing
- Recursive
- Divide and conquer algorithm

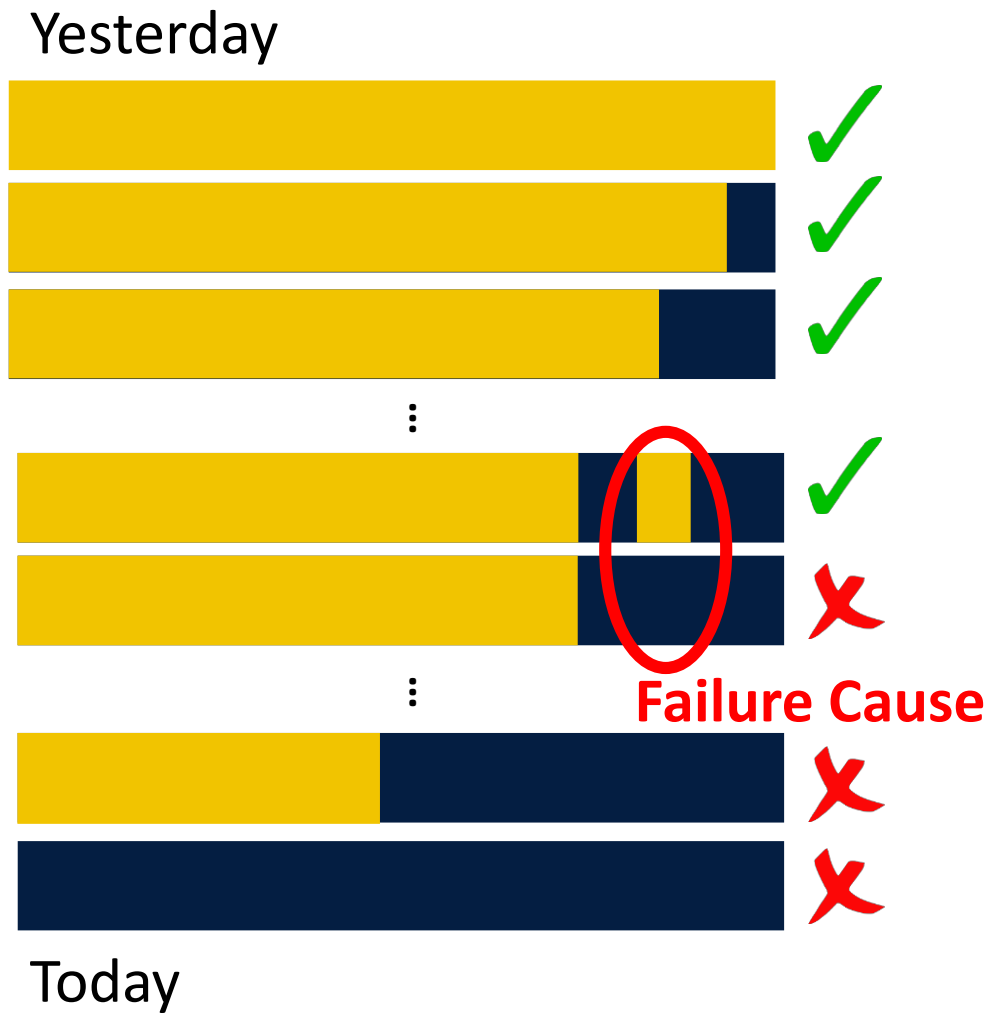


# Previous Techniques

- Linear search – not sufficient
  - Interference
  - Inconsistency
  - Granularity
- Debugging vs. isolating

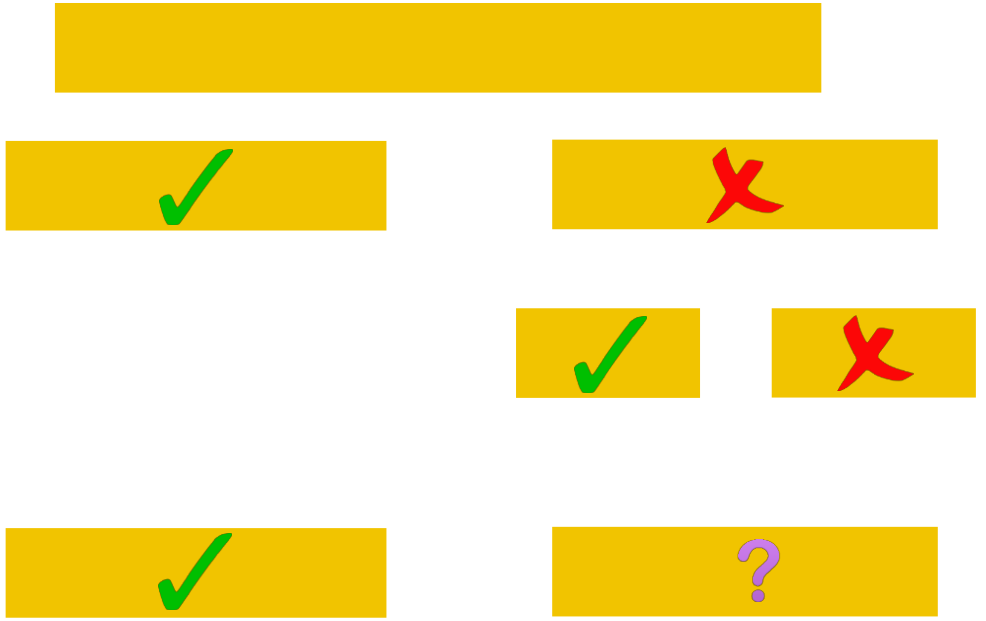
# Goal

- Minimal failure-inducing set
  - Least required subset to cause failure



# Binary Search

- Three outcomes:
  - Pass
  - Fail
  - Unresolved



# Configuration Properties

Certain properties allow for more efficiency

- **Monotony:** once failed, cannot un-fail
- **Unambiguity:** failure caused by one subset
- **Consistency:** repeatable

- Four scenarios

# Scenario 1:

- Hold all properties
- Possible Results:
  - Found in C1
  - Found in C2
  - Interference – both pass

Assume I = { 1, 2, 3, 4, 5, 6, **7**, 8}

| **Configuration** | **Results** |
|---|---|
| 1 2 3 4 5 6 **7** 8 | **FAIL** |
| 1 2 3 4 | PASS |
| 5 6 **7** 8 | **FAIL** |
| 5 6 | PASS |
| **7** 8 | **FAIL** |
| **7** | **FAIL** |

# Scenario 1:

- Hold all properties
- Possible Results:
  - Found in C1
  - Found in C2
  - Interference – both pass

Assume I = { 1, 2, **3**, 4, **5**, 6, **7**, 8}

| Configuration | | | | | | | | Results |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | **3** | 4 | **5** | 6 | **7** | 8 | **FAIL** |
| 1 | 2 | **3** | 4 | | | | | PASS |
| | | | | **5** | 6 | **7** | 8 | PASS |
| 1 | 2 | **3** | 4 | **5** | 6 | | | PASS |
| 1 | 2 | **3** | 4 | | | **7** | 8 | PASS |
| 1 | 2 | **3** | 4 | **5** | 6 | **7** | | **FAIL** |
| 1 | 2 | **3** | 4 | **5** | | **7** | | **FAIL** |
| 1 | 2 | | | **5** | | **7** | | PASS |
| | | **3** | 4 | **5** | | **7** | | **FAIL** |
| | | **3** | | **5** | | **7** | | **FAIL** |

# Scenarios 2 & 3

Scenario 2

- Configuration is ambiguous
- Find one, rerun, find another until complete

Scenario 3

- Not monotone
- “undoing” changes that cause pass

# Scenario 4

- Inconsistency
- Causes:
  - Integration failure
  - Construction failure
  - Execution failure

- Possible Results:
  - Found
  - Interference
  - Preference
  - Try again

# Scenario 4

## Preference

| Configuration | Results |
|---|---|
| 1 2 3 4 5 6 7 8 | FAIL |
| 1 2 3 4 | ? |
| 5 6 7 8 | PASS |
| 1 2 5 6 7 8 | PASS |

## Try Again

| Configuration | Results |
|---|---|
| 1 2 3 4 | ? |
| 5 6 7 8 | ? |
| 1 2 | ? |
| 3 4 | ? |

…

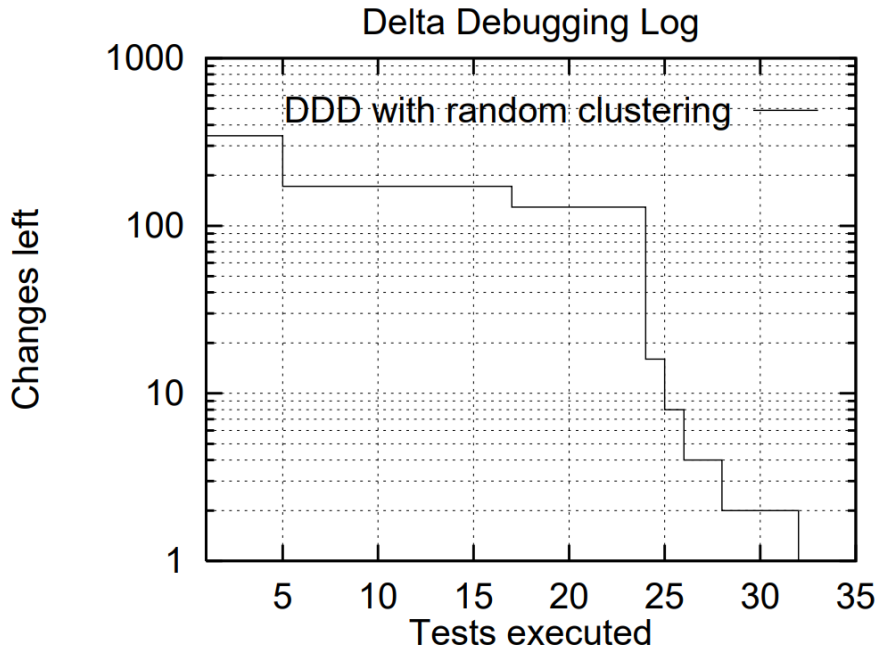
# Scenario 4

- Avoiding inconsistency
  - Grouping related changes
    - Location
    - Syntactic
  - Predicting test outcomes

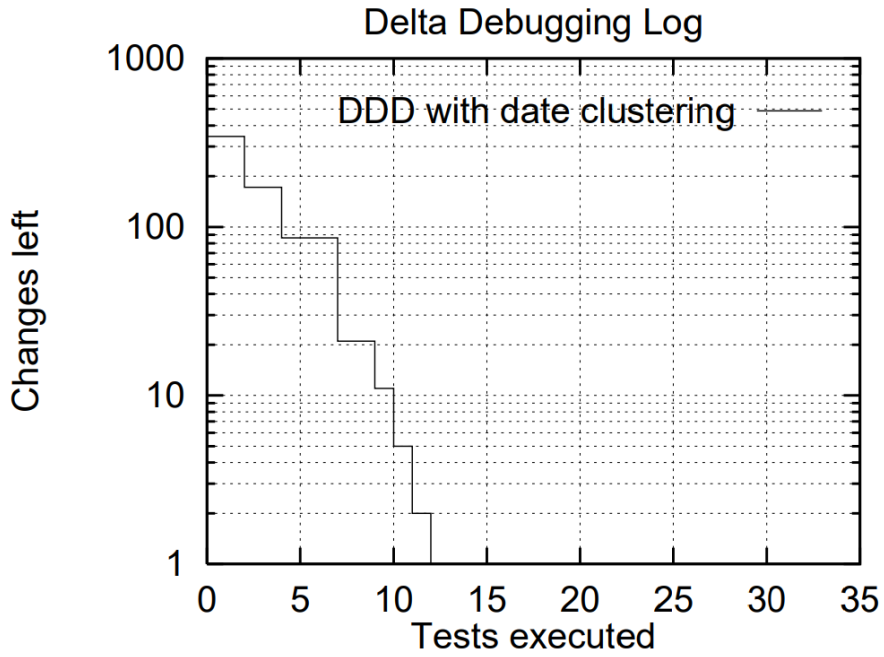
| Configuration | Results |
|---|---|
| 1 2 3 4 | PASS |
| 5 6 7 8 | Predict ? |
| 1 2 3 4 5 6 | PASS |
| 1 2 3 4 7 8 | Predict ? |
| 1 2 3 4 5 6 **7** | **FAIL** |

# Case Study 1

## Random Clustering

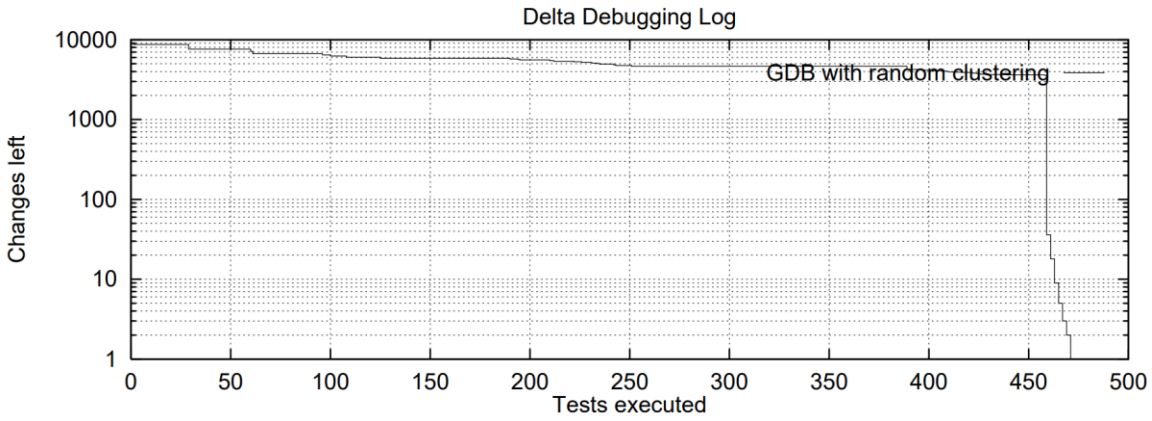



## Date Clustering

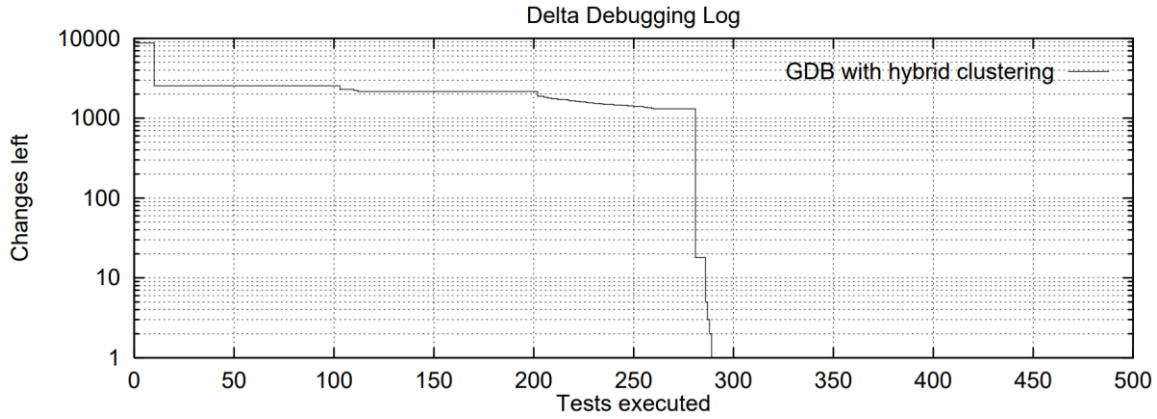


# Case Study 2

## Random Clustering




## Hybrid Clustering



# Using Magpie for request extraction and workload modeling

Paul Barham, Austin Donnelly, Rebecca Isaacs, Richard Mortier

Presented by; Won Park

# Motivation

- Ability to understand complex system behavior is useful
  - Performance analysis, debugging, etc.
- Still open questions / limitations in current work

# What is Magpie?

- Toolchain for automatically extracting system's workload under realistic operating conditions
- Provides ability to capture control path / resource demands of application requests
- Magpie operates both online / offline
  - Intention when designing was online

# Features of Magpie

- Collects events directly generated from kernel, middleware, application
- Ability to measure request's resource demands while discarding scheduling artefacts
  - OS multitasking / timesharing

# In short…

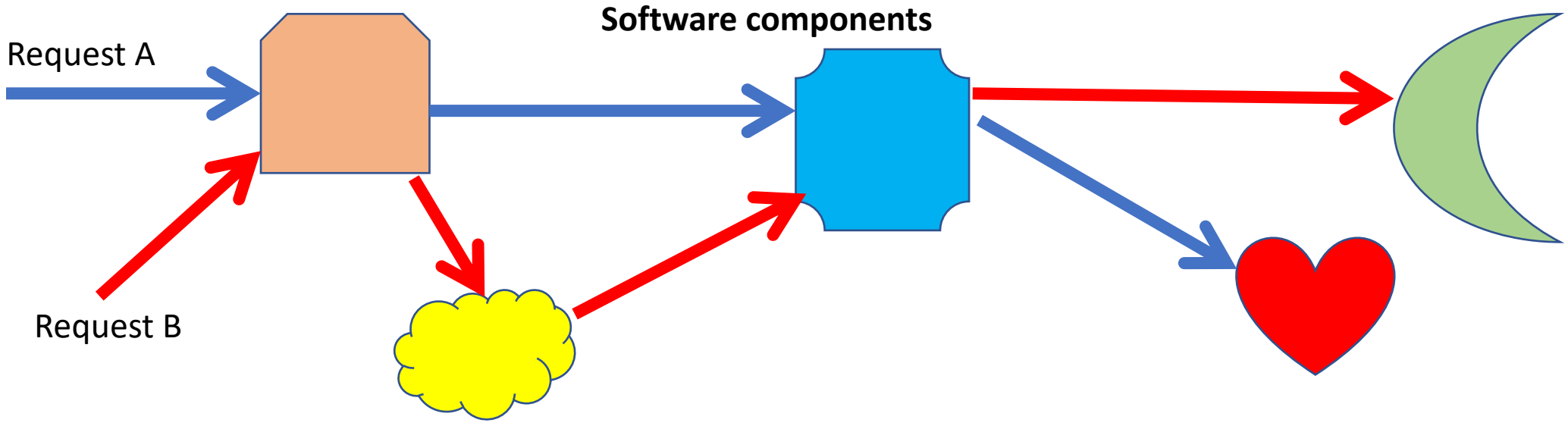
- Magpie provides picture of:
  - How request could have been serviced
  - How it actually was serviced
- Models are quicker + easier to understand
- Causes of problems often revealed by comparing Magpie trace to expected behavior

# In this talk..

- Background / Instrumentation
- Steps in the Toolchain
- Validation / Overhead Analysis
- Evaluation

# Background

- Workload of a system: different requests that take different paths




- Functional progress of request + resource consumption at every stage recorded

# Instrumentation Overview

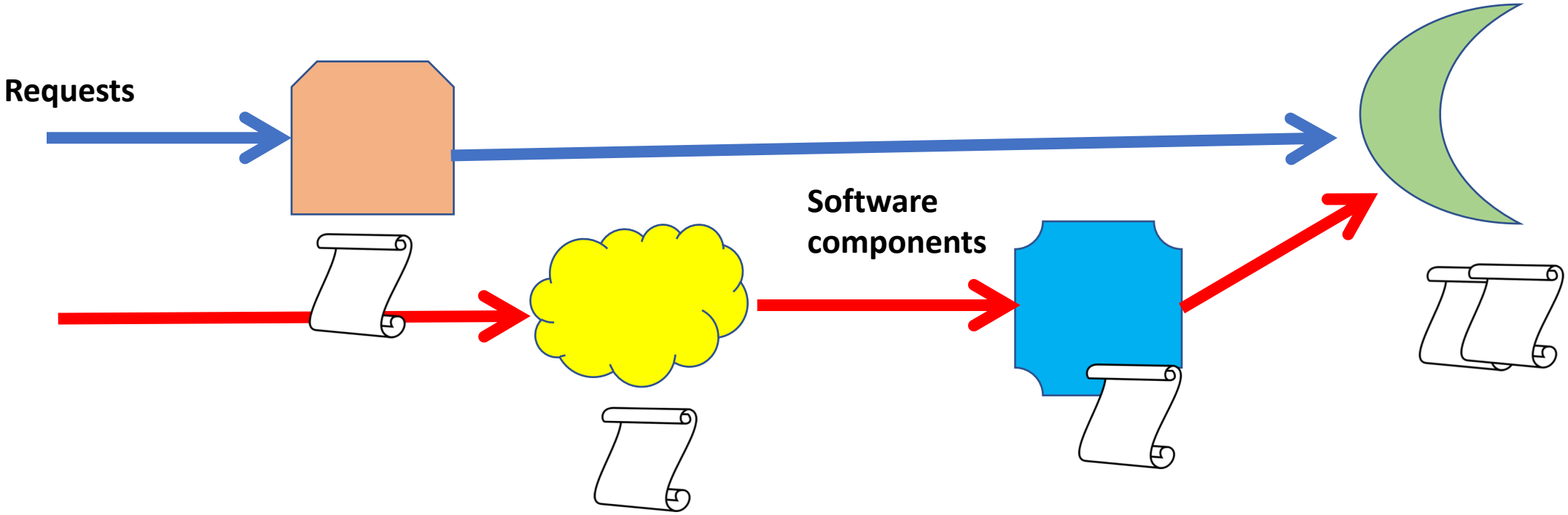
- Requirement: accurate accounting of resource usage
- Logical consequence: high precision timestamps needed

- Magpie’s solution: Processor cycle counter

# Event Tracing for Windows (ETW)

- Low overhead event logging infrastructure
- ETW event:
  - Timestamp
  - Provider / Event ID
  - Zero or more typed attributes

# Event Trace Production




Ordered but likely interleaved

# In this talk..

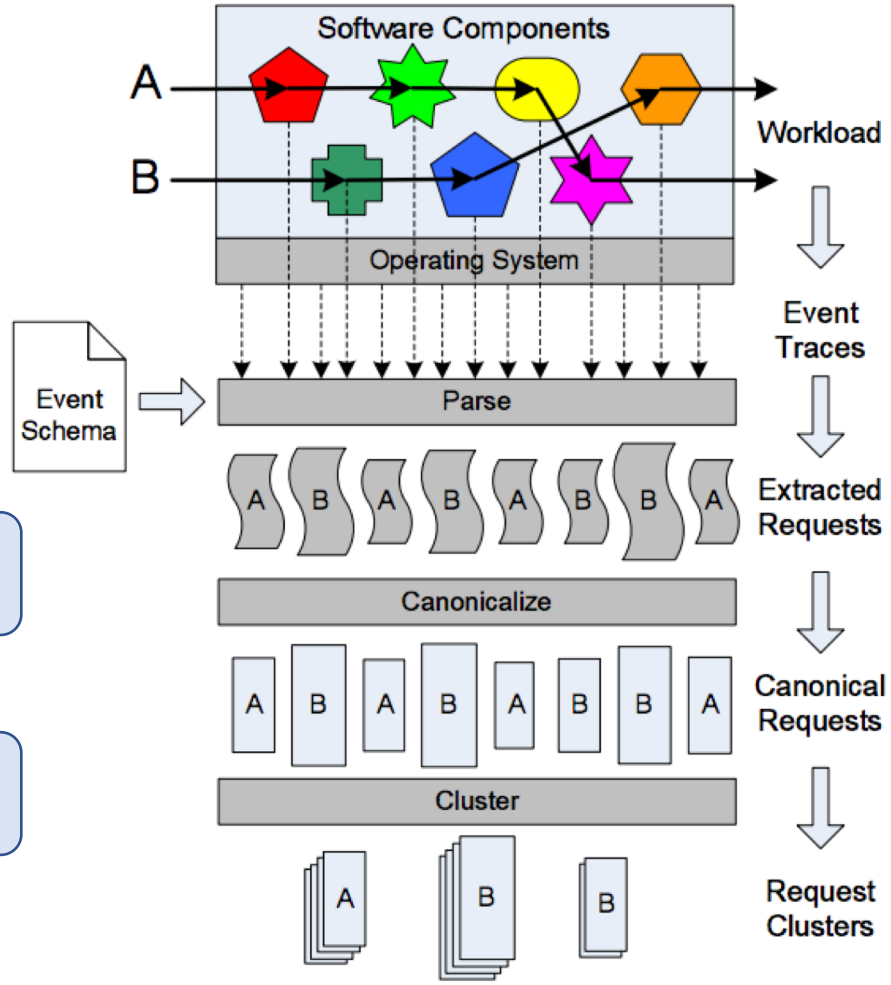
- Background / Instrumentation
- Steps in the Toolchain
- Validation / Overhead Analysis
- Evaluation

# Toolchain Overview

1. Parse requests

2. Canonicalize requests

3. Cluster requests

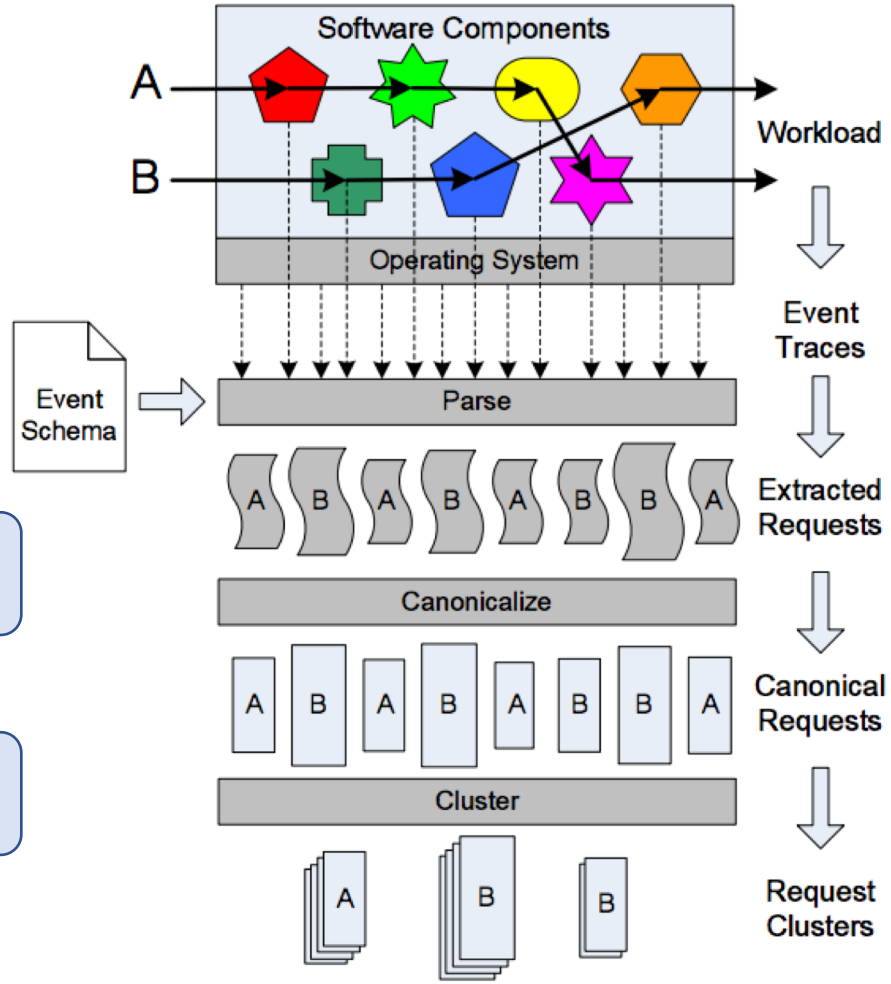



TODO: during practice talk, think about where to insert graphics talkinga bout the software componetns

1. Parse requests

2. Canonicalize requests

3. Cluster requests



# Request Parser

- Responsible for extracting requests from event logs

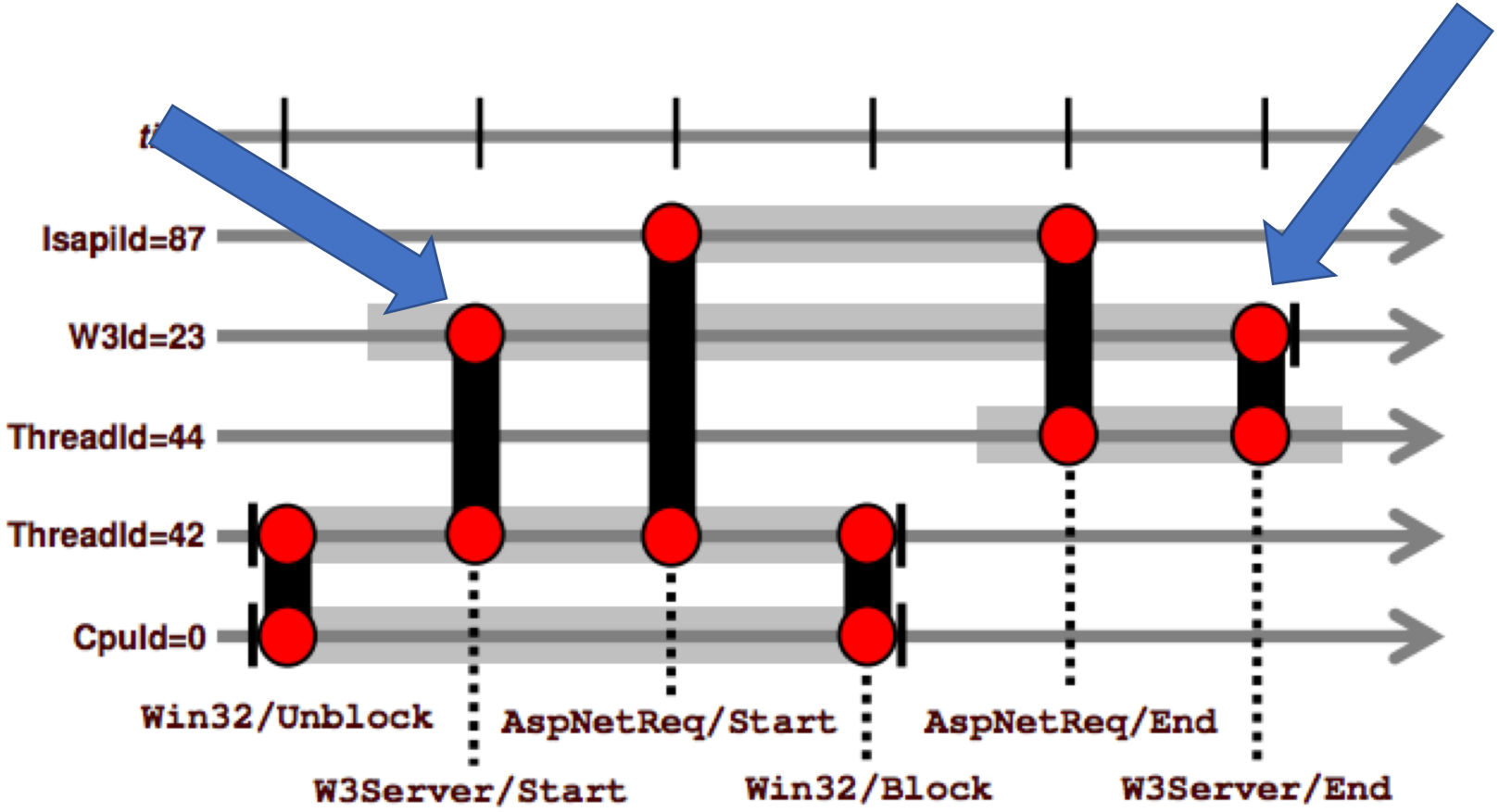

- Speculatively builds up sets of related events

# Event Schema

- Specifies which attribute connects events to other events
- Application specific
  - Request parser itself makes no built-in assumptions

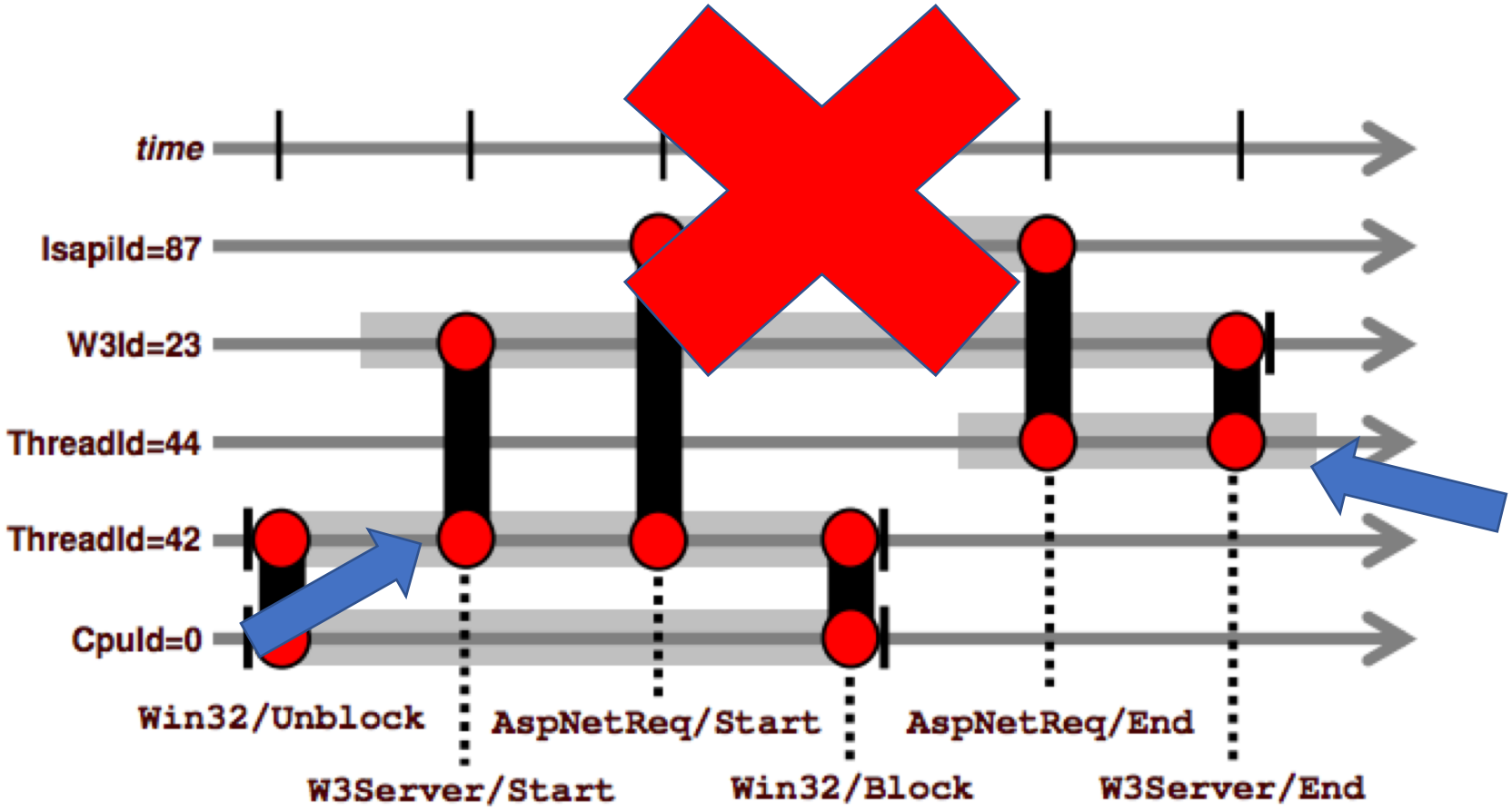
```
Event (“W3Server”, “Start”);
Attribute (“TId”, BIND_BASIC, 0);
Attribute (“W3Id”, BIND_BASIC, 0);
```

```
Event (“W3Server”, “End”);
Attribute (“TId”, BIND_BASIC, 0);
Attribute (“W3Id”, BIND_STOP, 0);
```



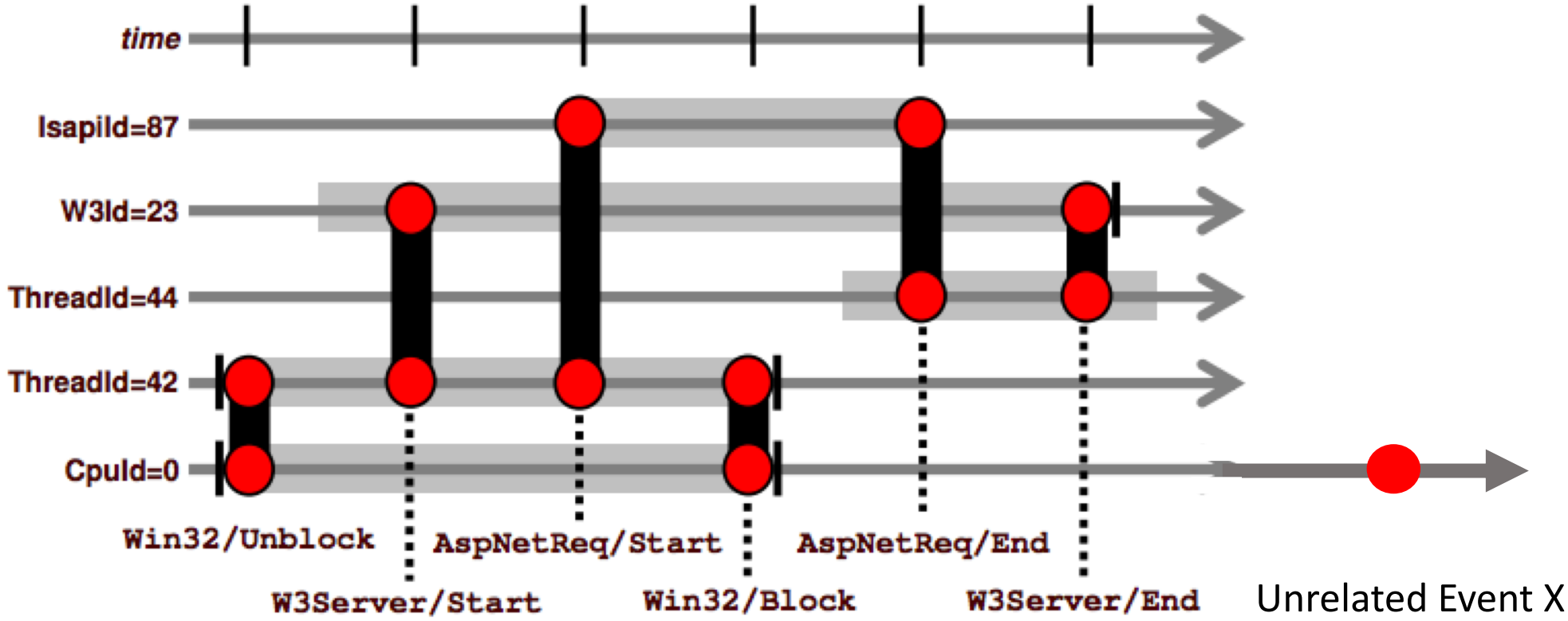

Event (“W3Server”, “Start”);
Attribute (“TId”, BIND_BASIC, 0);
Attribute (“W3Id”, BIND_BASIC, 0);

Event (“W3Server”, “End”);
Attribute (“TId”, BIND_BASIC, 0);
Attribute (“W3Id”, BIND_STOP, 0);




Event (“W3Server”, “Start”);
Attribute (“TId”, BIND_BASIC, 0);
Attribute (“W3Id”, BIND_BASIC, 0);

Event (“W3Server”, “End”);
Attribute (“TId”, BIND_BASIC, 0);
Attribute (“W3Id”, BIND_STOP, 0);



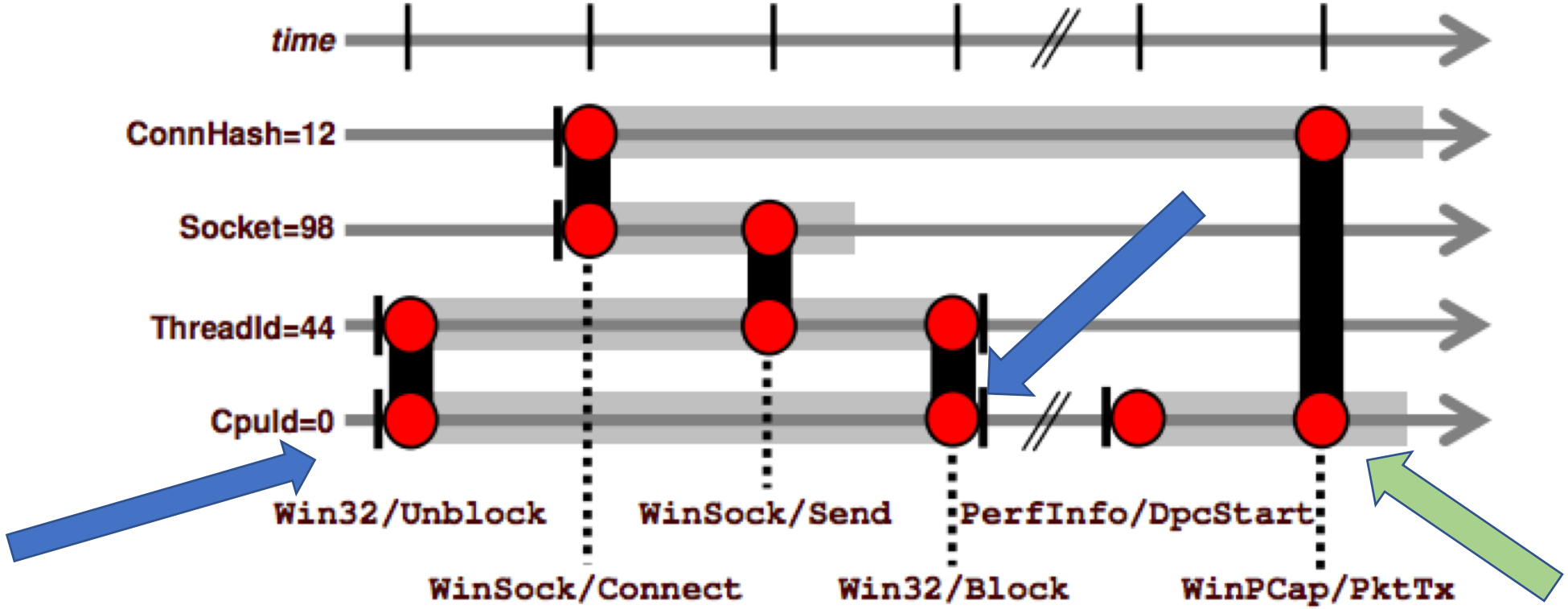

What prevents X from being joined with other unrelated events?

# Temporal Joins

- Valid interval: period in which we know one resource is working exclusively on one request

- During a valid interval: events joined together as usual
- Once valid interval closed: no more events added to event via the specified pair

- This data is built into schema




New events not added via that attribute pair

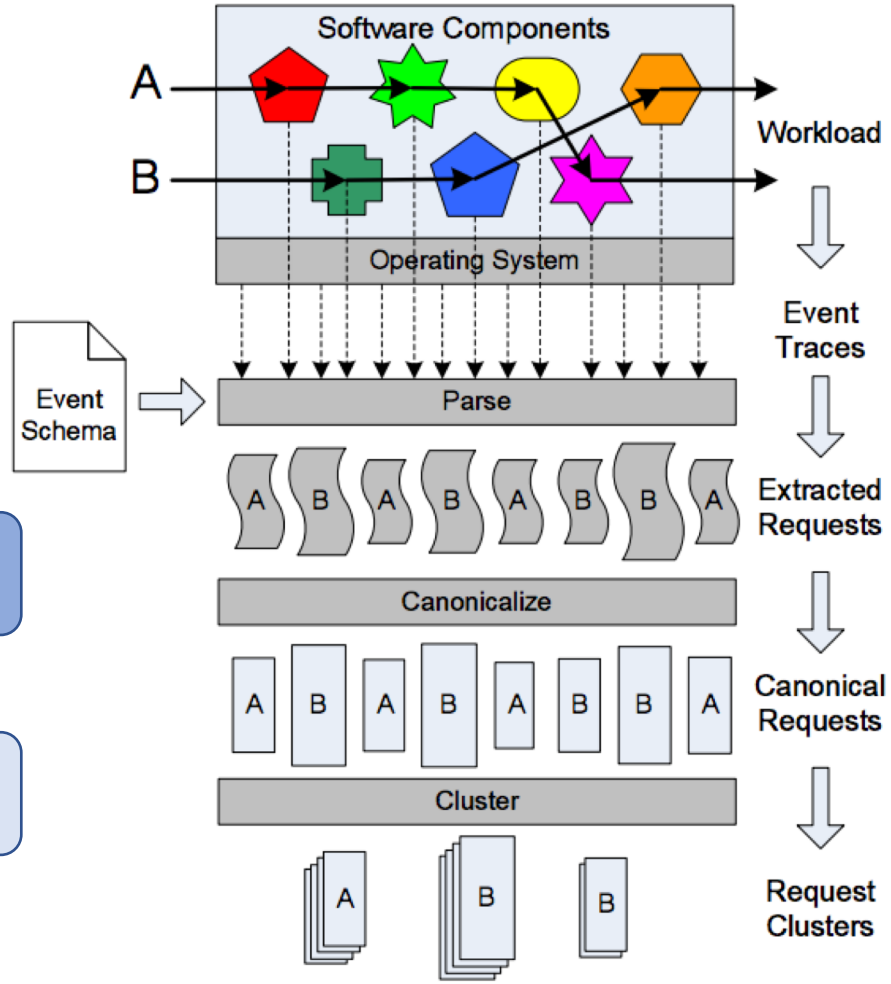
Event (“Win32”, “Unblock”);
Attribute (“CpuID”, BIND_START, 0);
Attribute (“TId”, BIND_START, 0);

Event (“Win32”, “Block”);
Attribute (“CpuID”, BIND_STOP, 0);
Attribute (“TId”, BIND_STOP, 0);

1. Parse requests

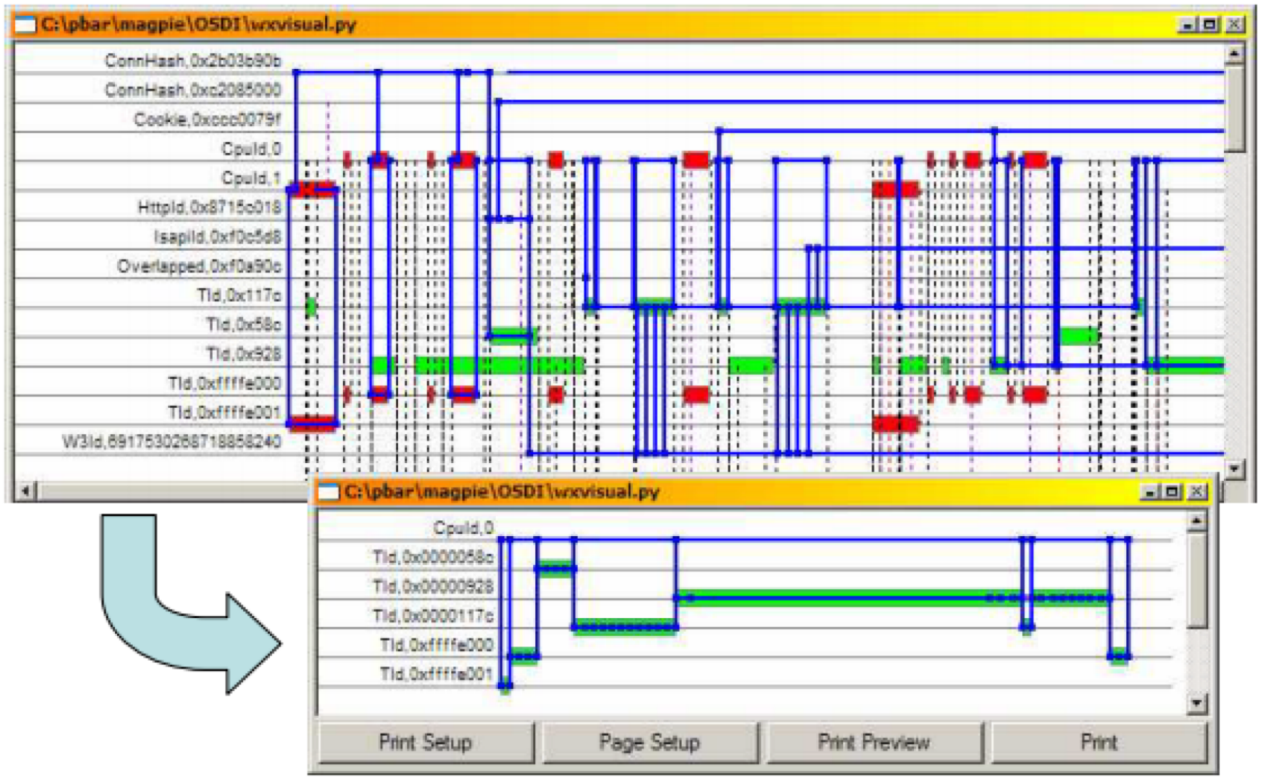
2. Canonicalize requests

3. Cluster requests



# Canonicalization

- Problem: we may want to know requests as sequence of demands
  - Not about how it was actually serviced
- Use cause ordering annotations to create canonical version of each request




"Clean out the request"

# How to handle synchronization?

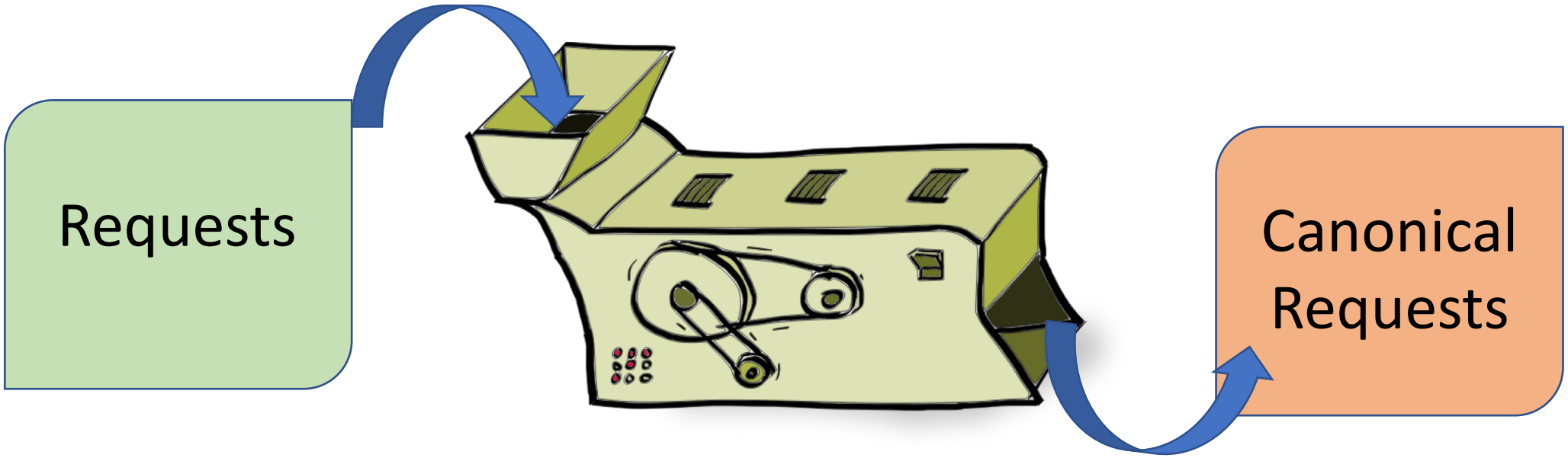
- Need to recover threading structure within request
  - E.g. identify when thread blocks or causes another to unblock
- Magpie has mechanism to explicitly insert causal dependencies into parsed event graphs

# How to handle requests on multiple machines?

- Run request parser once for each machine
- Request fragments canonicalized
- Run offline tool to combine canonical fragments by connecting synchronization events

# In short…

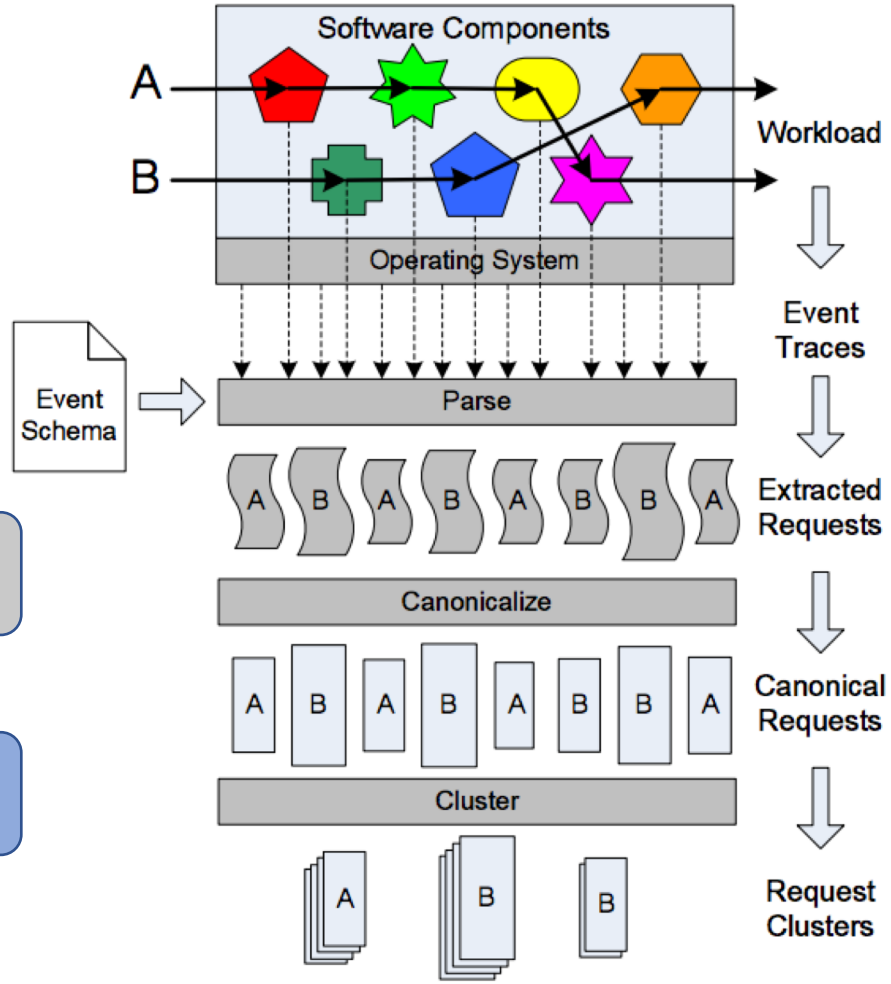
- Create canonicalized version of requests
  - Concatenate resource demands between synchronization points



1. Parse requests

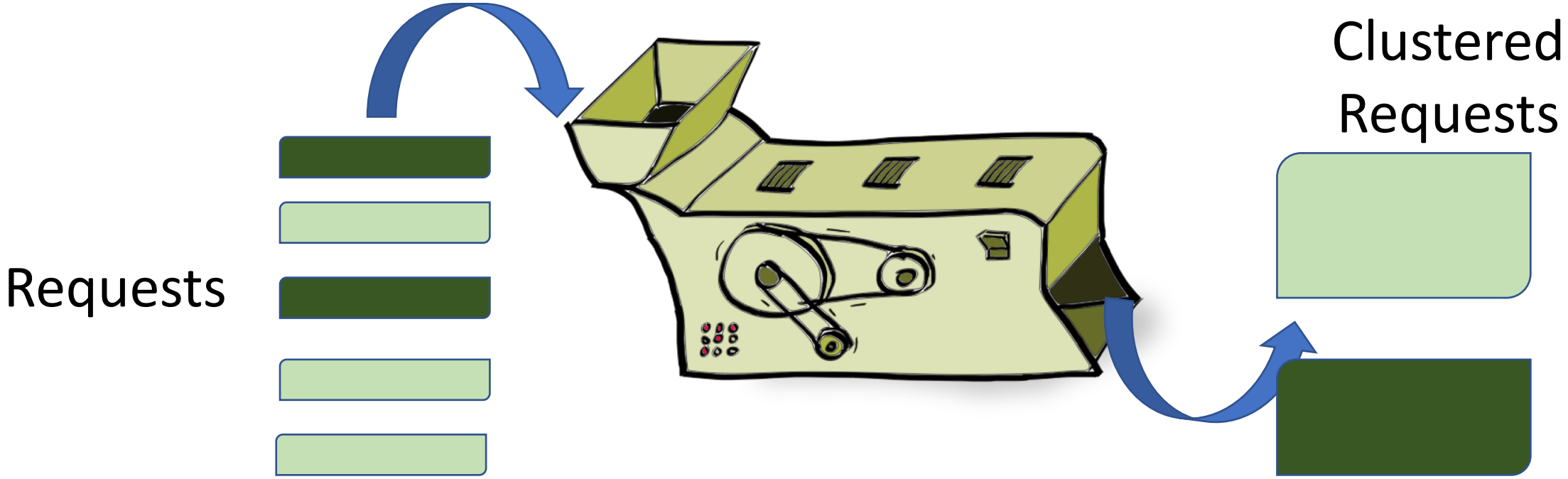
2. Canonicalize requests

3. Cluster requests



# Behavioral Clustering

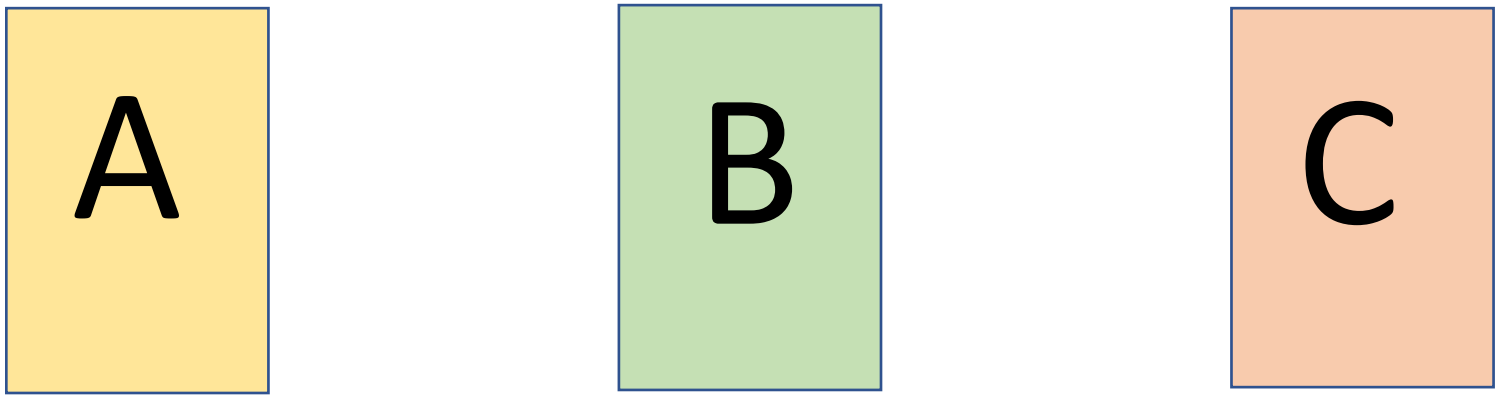
- Groups together requests with similar behavior
- Use single incremental clustering algorithm



# Behavioral Clustering Steps

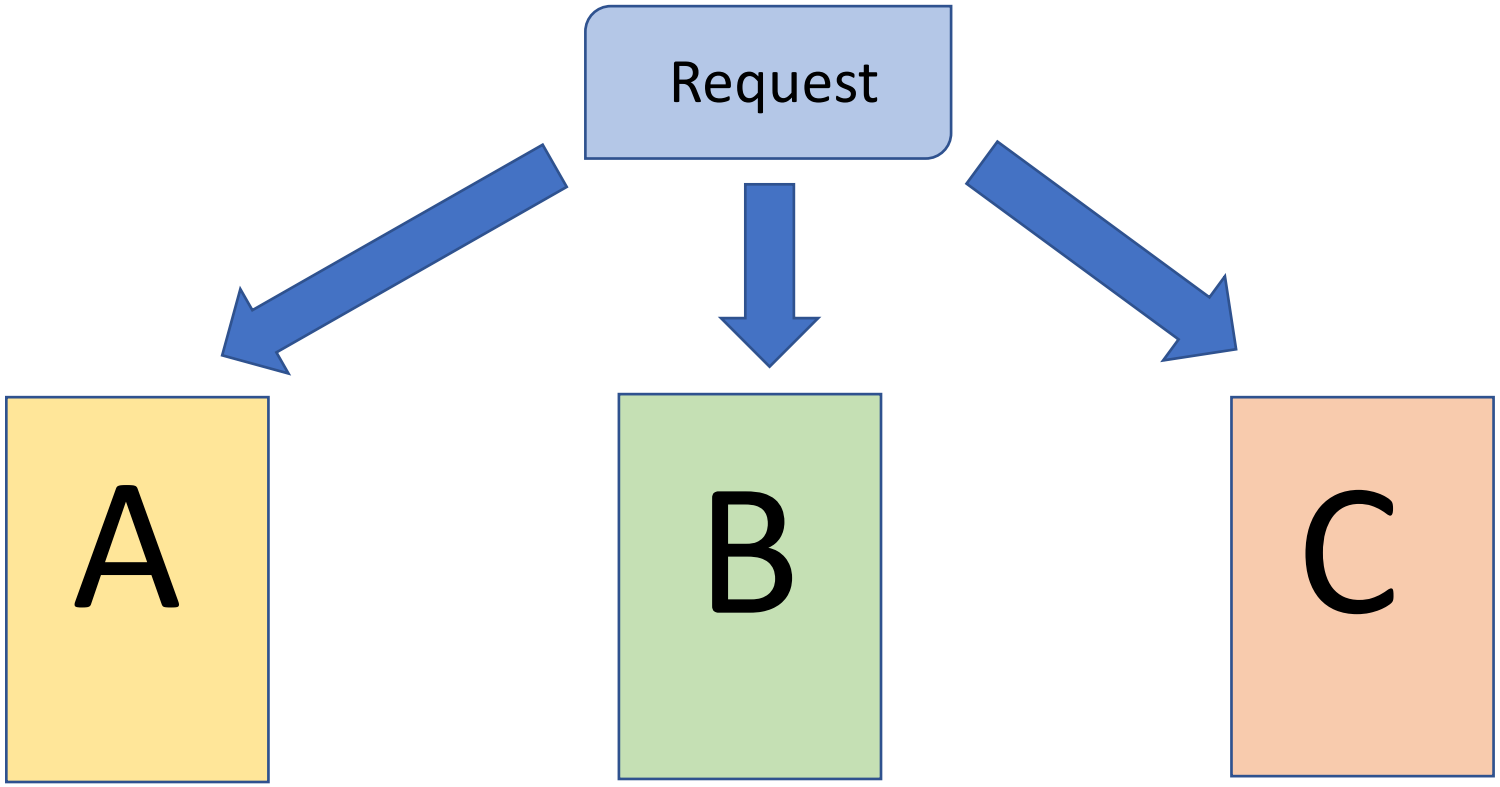
Clusterer maintains set of active clusters

Use representative centroid



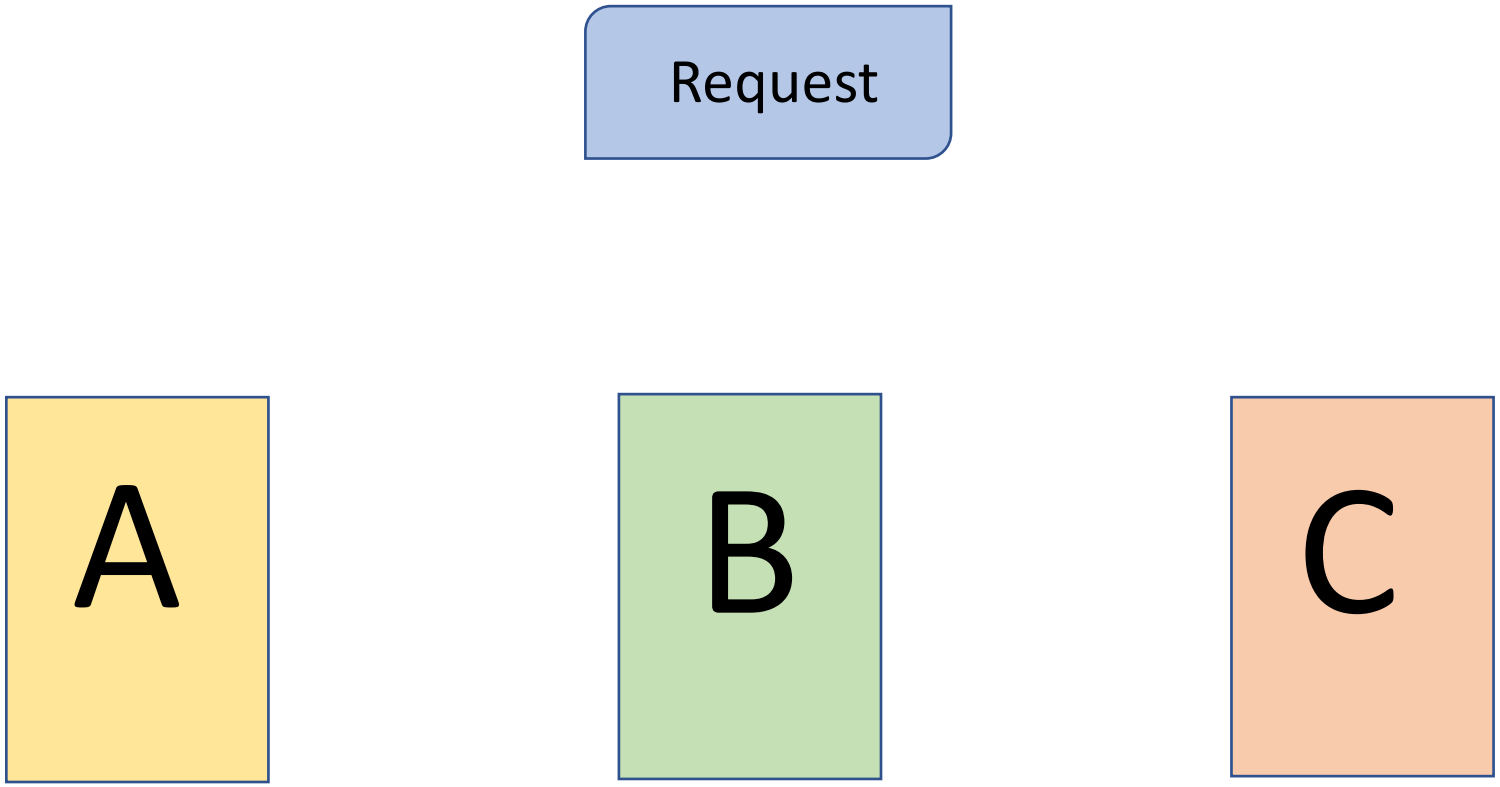
# Behavioral Clustering Steps

For each request, compute string-edit-distance to centroid(s)



# Behavioral Clustering Steps

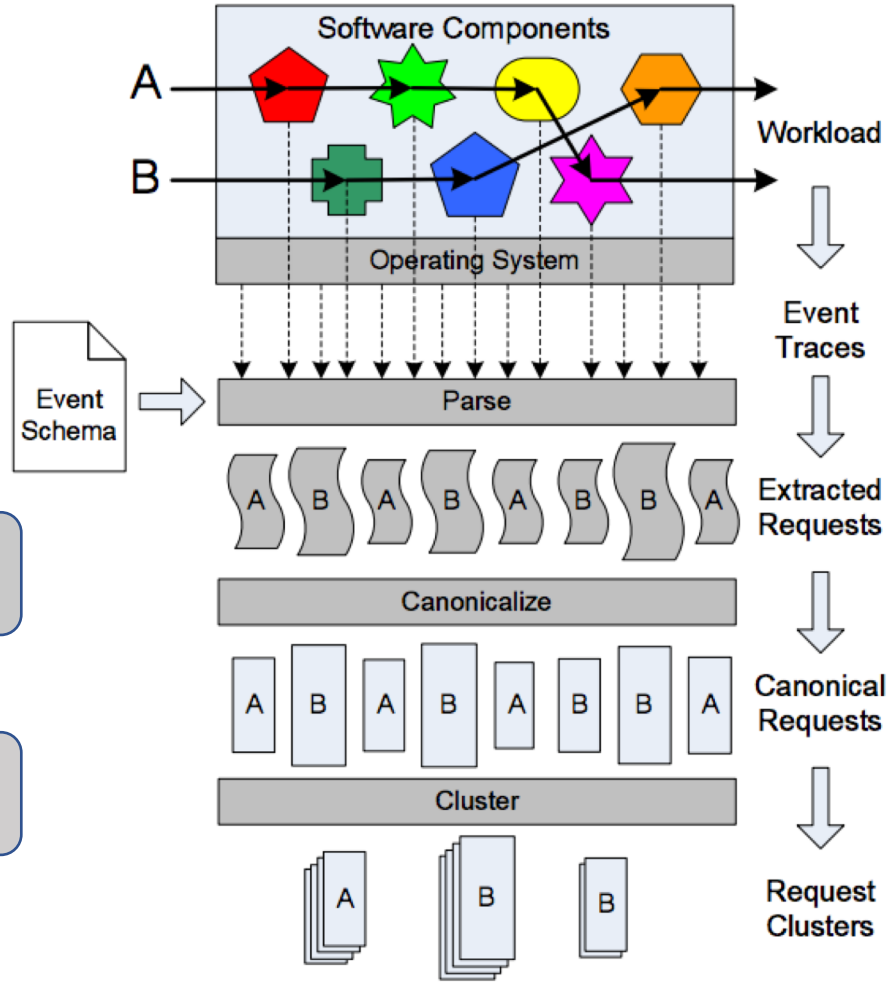
Add to cluster with smallest distance or create new cluster



1. Parse requests

2. Canonicalize requests

3. Cluster requests



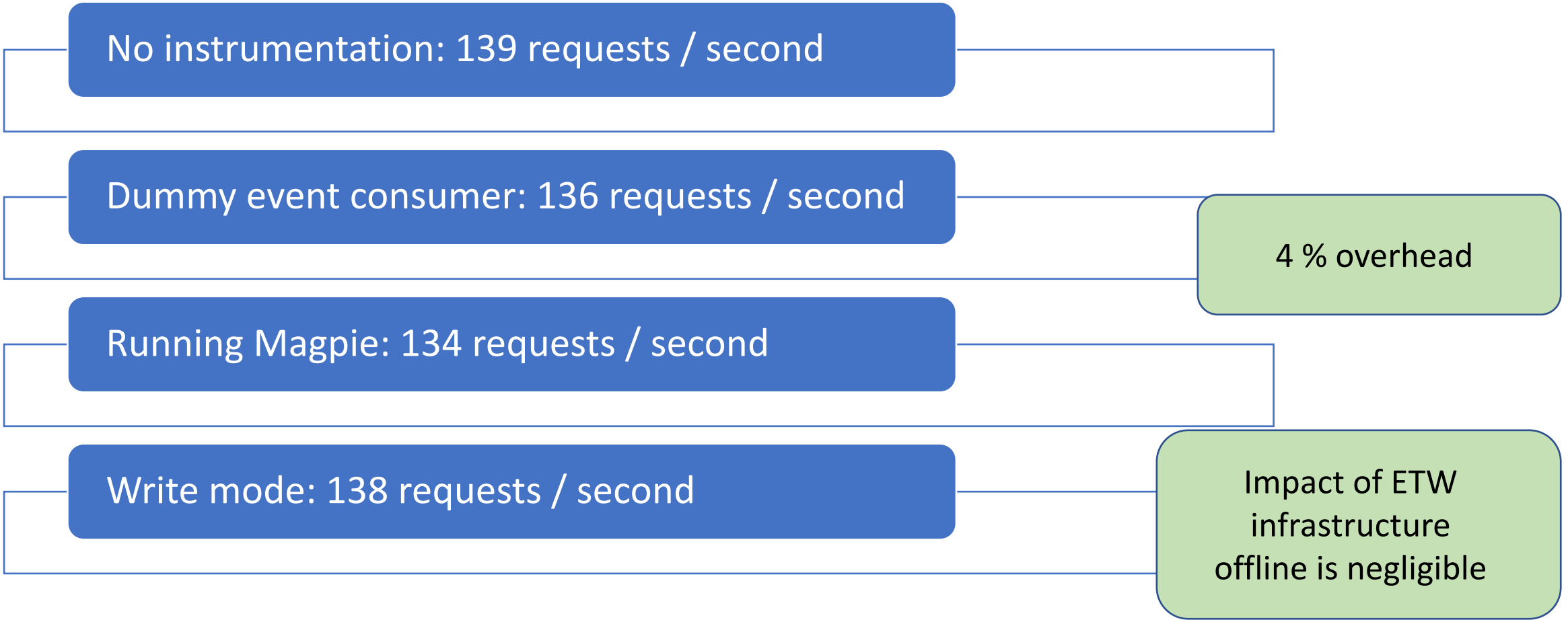
# In this talk..

- Background / Instrumentation
- Steps in the Toolchain
- Validation / Overhead Analysis
- Evaluation

# Overhead

- Assessed performance impact of event tracing / parsing using web stress-test
- CPU bound workload that generated active HTML content that saturated CPU
- Counted # HTTP requests served over two minute interval

# Overhead


No instrumentation: 139 requests / second
Dummy event consumer: 136 requests / second
4 % overhead
Running Magpie: 134 requests / second
Write mode: 138 requests / second
Impact of ETW infrastructure offline is negligible

# Validation

- Evaluated accuracy of Magpie’s models using traces from synthetic workload
- Calibrated to check workload output was same as input

# Validation: Processor Time

- Used two types of requests:

| Type | URL | Resource |
|---|---|---|
| A | longspin.aspx | 1 thread spin CPU 30ms |
| B | shortspin.aspx | 1 thread spin CPU 15ms |

- Combination of:
  - Serialized vs. Concurrent (from 5 clients)
  - Single Variety or Mixed

| Exp no. | Workload / # clients | | Single? | Accnt CPU % | Clusters found | Dia. | Min. Sep. | Model Error |
|---|---|---|---|---|---|---|---|---|
| 1.1 | 500 A | x1 | Single | 96.6% | 499 | 0.6 | 130 | 2.4% |
| | | | | | 1 | 0 | 130 | |
| 1.2 | | | | 94.0% | 500 | | 0.0 | |
| 1.3 | | | | 96.6% | 49 | | 21.7 | |
| | | | | | 1 | | 21.7 | |
| | | | | | 1 | | 13.1 | |
| 1.4 | 500 B | x5 | | | 500 | | 0.0 | |
| 1.5 | 500 (A/B) | x1 | | | 266A | | 8.5 | |
| | | | | | 234B | | .5 | |
| 1.6 | 500 (A/B) | x5 | | | 244A | | .0 | 2.0% |
| | | | | | 254B | | .0 | |
| | | | | | 1A | | 33.2 | |
| | | | | | 1B | 0.0 | 103 | |

**Accnt CPU%:** fraction of CPU consumed by relevant process

**Diameter**: Avg. cluster diameter

**Model error**: Diff. in resource consumption between derived workload model and parsed requests

**Clusters found**: # of requests found in each cluster

**Min sep**: distance to centroid of nearest cluster

| Exp no. | Workload / # clients | | Single? | Accnt CPU % | Clusters found | Dia. | Min. Sep. | Model Error |
|---|---|---|---|---|---|---|---|---|
| 1.1 | 500 A | x1 | Single | 96.6% | 499 | 0.6 | 130 | 2.4% |
| | | | | | 1 | 0 | 130 | |
| 1.2 | 500 B | x1 | Single | 94.0% | 500 | 0.4 | 0.0 | 2.1% |
| 1.3 | 500 A | x5 | Conc. | 96.6% | 498 | 1.2 | 21.7 | 3.2% |
| | | | | | 1 | 0.0 | 21.7 | |
| | | | | | 1 | 0.0 | 13.1 | |
| 1.4 | 500 B | x5 | Conc. | 94.6% | 500 | 1.2 | 0.0 | 2.8% |
| 1.5 | 500 (A/B) | x1 | Mixed | 95.9% | 266A | 1.1 | 8.5 | 0.9% |
| | | | | | 234B | 1.7 | 8.5 | |
| 1.6 | 500 (A/B) | x5 | Mixed | 95.9% | 244A | 1.2 | 8.0 | 2.0% |
| | | | | | 254B | 1.2 | 8.0 | |
| | | | | | 1A | 0.0 | 33.2 | |
| | | | | | 1B | 0.0 | 103 | |

| Exp no. | Workload / # clients | | Single? | Accnt CPU % | Clusters found | Dia. | Min. Sep. | Model Error |
|---|---|---|---|---|---|---|---|---|
| 1.1 | 500 A | x1 | Single | 96.6% | 499 | 0.6 | 130 | **2.4%** |
| 1.2 | 500 B | x1 | Single | 94. | | | | **2.1%** |
| 1.3 | 500 A | x5 | Conc. | 96. | | | | **3.2%** |
| 1.4 | 500 B | x5 | Conc. | 94. | | | | **2.8%** |
| 1.5 | 500 (A/B) | x1 | Mixed | 95. | | | | **0.9%** |
| 1.6 | 500 (A/B) | x5 | Mixed | 95. | | | | **2.0%** |
| | | | | | 1A | 0.0 | 33.2 | |
| | | | | | 1B | 0.0 | 103 | |

In all cases, cluster size / representatives can be used to represent workload to within 3.2%

# Magpie: anomaly detection

| Exp no. | Workload / # clients | | Single? | Accnt CPU % | Clusters found | Dia. | Min. Sep. | Model Error |
|---|---|---|---|---|---|---|---|---|
| 1.1 | 500 A | x1 | Single | 96.6% | 499 | 0.6 | 130 | 2.4% |
| | | | | | 1 | 0 | 130 | |
| 1.2 | 500 B | x1 | Single | 94.0% | 500 | 0.4 | 0.0 | 2.1% |
| 1.3 | 500 A | x5 | Conc. | 96.6% | 498 | 1.2 | 21.7 | 3.2% |
| | | | | | 1 | 0.0 | 21.7 | |
| | | | | | 1 | 0.0 | 13.1 | |
| 1.4 | 500 B | x5 | Conc. | 94.6% | 500 | 1.2 | 0.0 | 2.8% |
| 1.5 | 500 | x1 | Mixed | 95.9% | 266A | 1.1 | 8.5 | 0.9% |
| | | | | | 234B | 1.7 | 8.5 | |
| | | | | %| 244A | 1.2 | 8.0 | 2.0% |
| | | | | | 254B | 1.2 | 8.0 | |
| | | | | | 1A | 0.0 | 33.2 | |
| | | | | | 1B | 0.0 | 103 | |

Discovered that driver was calling some function with improper args

# Concurrency

How well is Magpie able to capture differences in concurrency structure?

| Type | URL | Resource |
|---|---|---|
| A | longspin.aspx | 1 thread spin CPU 30ms |
| B | shortspin.aspx | 1 thread spin CPU 15ms |
| E | parspin.aspx | 2 threads spin CPU 15ms concurrently |

| Exp No | Workload/Clients | | Accnt CPU % | Clusters Found | Dia. | Min. Sep | Model Error |
|---|---|---|---|---|---|---|---|
| 3.3 | 500 (B/E) | x1 | 95.8% | 267E | 0.5 | 8.7 | 0.6% |
| | | | | 232B | 0.4 | 8.8 | |
| | | | | 1E | 0.0 | 34.7 | |
| 3.5 | 500 (A/B/E) | x1 | 96.1% | 172E | 0.5 | 8.9 | 3.0% |
| | | | | 168A | 0.6 | 8.1 | |
| | | | | 160B | 0.5 | 8.1 | |
| 3.6 | 500 (A/B/E) | x5 | 96.2% | 187E | 0.9 | 0.0 | 0.3% |
| | | | | 152B | 0.6 | 8.6 | |
| | | | | 160A | 1.2 | 8.6 | |
| | | | | 1A | 0.0 | 130 | |

| Exp No | Workload/Clients | | Accnt CPU % | Clusters Found | Dia. | Min. Sep | Model Error |
|---|---|---|---|---|---|---|---|
| 3.3 | 500 (B/E) | x1 | 95.8% | 267E | 0.5 | 8.7 | 0.6% |
| | | | | 232B | 0.4 | 8.8 | |
| | | | | 1E | 0.0 | 34.7 | |
| 3.5 | 500 (A/B/E) | x1 | 96.1% | 172E | 0.5 | 8.9 | 3.0% |
| | | | | 168A | 0.6 | 8.1 | |
| | | | | 160B | 0.5 | 8.1 | |
| 3.6 | 500 (A/B/E) | x5 | 96.2% | 187E | 0.9 | 0.0 | 0.3% |
| | | | | 152B | 0.6 | 8.6 | |
| | | | | 160A | 1.2 | 8.6 | |
| | | | | 1A | 0.0 | 130 | |

Distance metric / clustering algo capable of separating requests that differ only in internal concurrency structure

# In this talk..

- Background / Instrumentation
- Steps in the Toolchain
- Validation / Overhead Analysis
- Evaluation

# Small two-tier website: Duwamish

- Setup: 2 machine system running Duwamish bookstore
  - Packed with generated data in accordance with volume / size

| Exp no. | Workload | Accnt CPU % | Clusters Found | Max Dia | Min Sep |
|---|---|---|---|---|---|
| 5.1 | Book (B) | 82.8% | WEB: 404 + 92 + 4 | 0.8 | 3.6 |
| | | 96.1% | SQL: 500 | 0.5 | 0.0 |
| | | | E2E: 408 + 92 | 1.2 | 5.1 |
| 5.2 | Logon (L) | 86.5% | WEB: 492 + 2 + 1 | 1.3 | 7.3 |
| 5.3 | Categories (C) | 95.2% | WEB: 445 + 26 + 29 | 2.3 | 14.9 |
| | | 97.3% | SQL: 1499 + 1 | 1.3 | 16.4 |
| | | | E2E: 444 + 26 +29 + 1 | 2.6 | 17.9 |
| 5.4 | Mixed (B/L/C) | 91.7% | WEB: (138L + 1B) + 220B + 61C+ 21C + 24C + 10 * 25C | 4.9 | 5.9 |
| | | 96.7% | SQL: ($141C_1$ + 122 $C_2$ + 141 $C_3$ + 221B) + 9 $C_2$ + $10C_2$ | 2.3 | 19.7 |
| | | | E2E:(138L + 1B) + 220B + 55C + 47CF + 14C + 10 * 25C | 5.1 | 8.2 |

| Exp no. | Workload | Accnt CPU % | Clusters Found | Max Dia | Min Sep |
|---|---|---|---|---|---|
| 5.1 | Book (B) | 82.8% | WEB: 404 + 92 + 4 | 0.8 | 3.6 |
| | | 96.1% | SQL: 500 | 0.5 | 0.0 |
| | | | E2E: 408 + 92 | 1.2 | 5.1 |
| 5.2 | Logon (L) | 86.5% | WEB: 492 + 2 + 1 | 1.3 | 7.3 |
| 5.3 | Categories (C) | 95.2% | WEB: 445 + 26 + 29 | 2.3 | 14.9 |
| | | 97.3% | SQL: 1499 + 1 | 1.3 | 16.4 |
| | | | E2E: 444 + 26 +29 + 1 | 2.6 | 17.9 |
| | | 1.7% | WEB: (138L + 1B) + 220B + 61C+ 21C + 24C + 10 * 25C | 4.9 | 5.9 |
| | | 6.7% | SQL: ($141C_1$ + 122 $C_2$ + 141 $C_3$ + 221B) + 9 $C_2$ + $10C_2$ | 2.3 | 19.7 |
| | | | E2E:(138L + 1B) + 220B + 55C + 47CF + 14C + 10 * 25C | 5.1 | 8.2 |

Groups for Book / Logon much tighter than those for Categories

Workload model based on observed behavior > workload model based on URL measurements

# Conclusion

- Magpie is a toolchain that takes events generated by OS, middleware, application components
- Able to output a model of how requests were serviced and how it could have been serviced
- Validated approach against synthetic workloads

# Questions?