Diagnosing performance changes by comparing request flows

Raja Sambasivan

Alice Zheng★, Michael De Rosa✚, Elie Krevat,
Spencer Whitman, Michael Stroucken,
William Wang, Lianghong Xu, Greg Ganger

Carnegie Mellon Microsoft Research★ Google✚

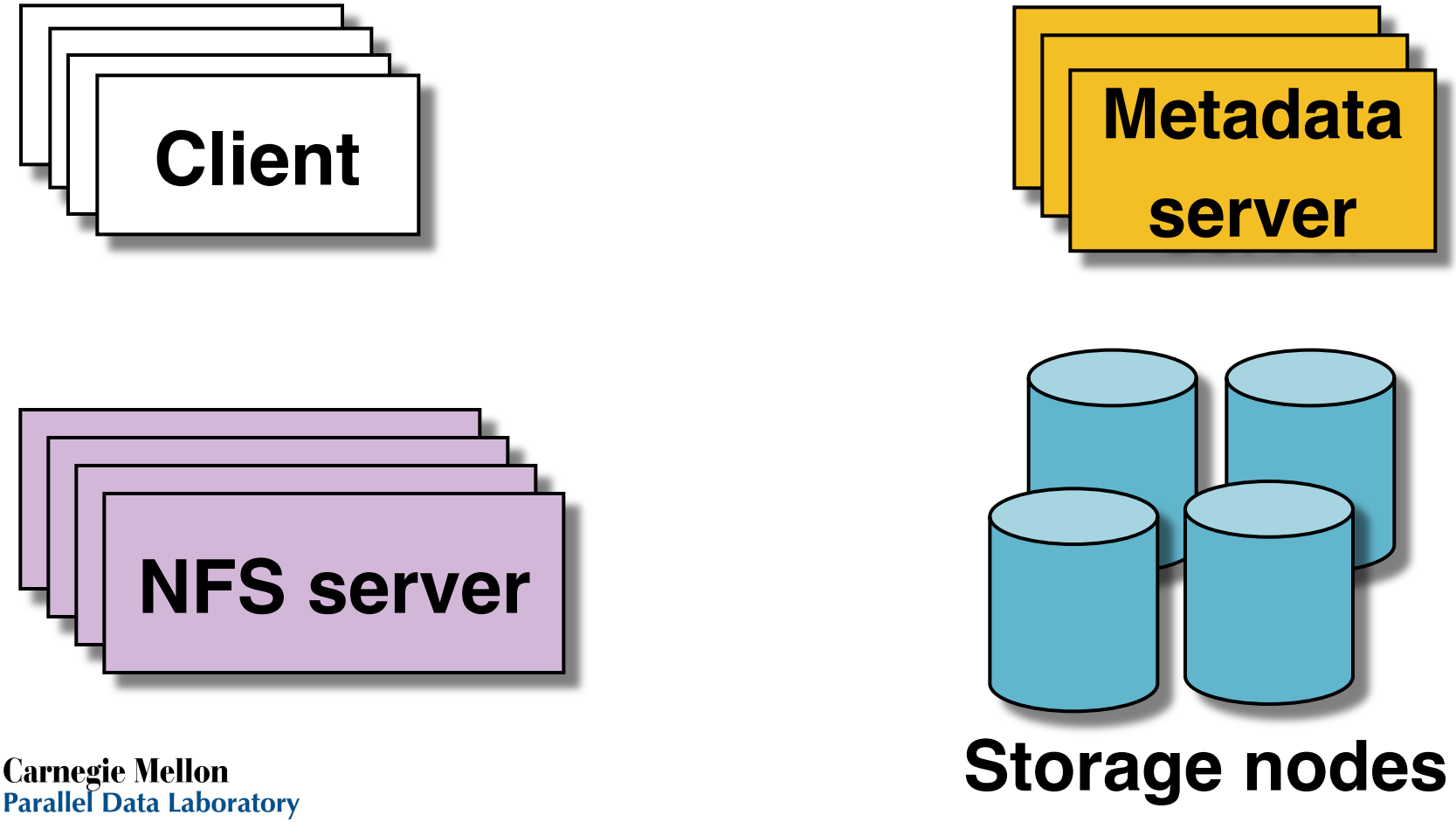
Carnegie Mellon
Parallel Data Laboratory

Perf. diagnosis in distributed systems

- Very difficult and time consuming
  - Root cause could be in any component

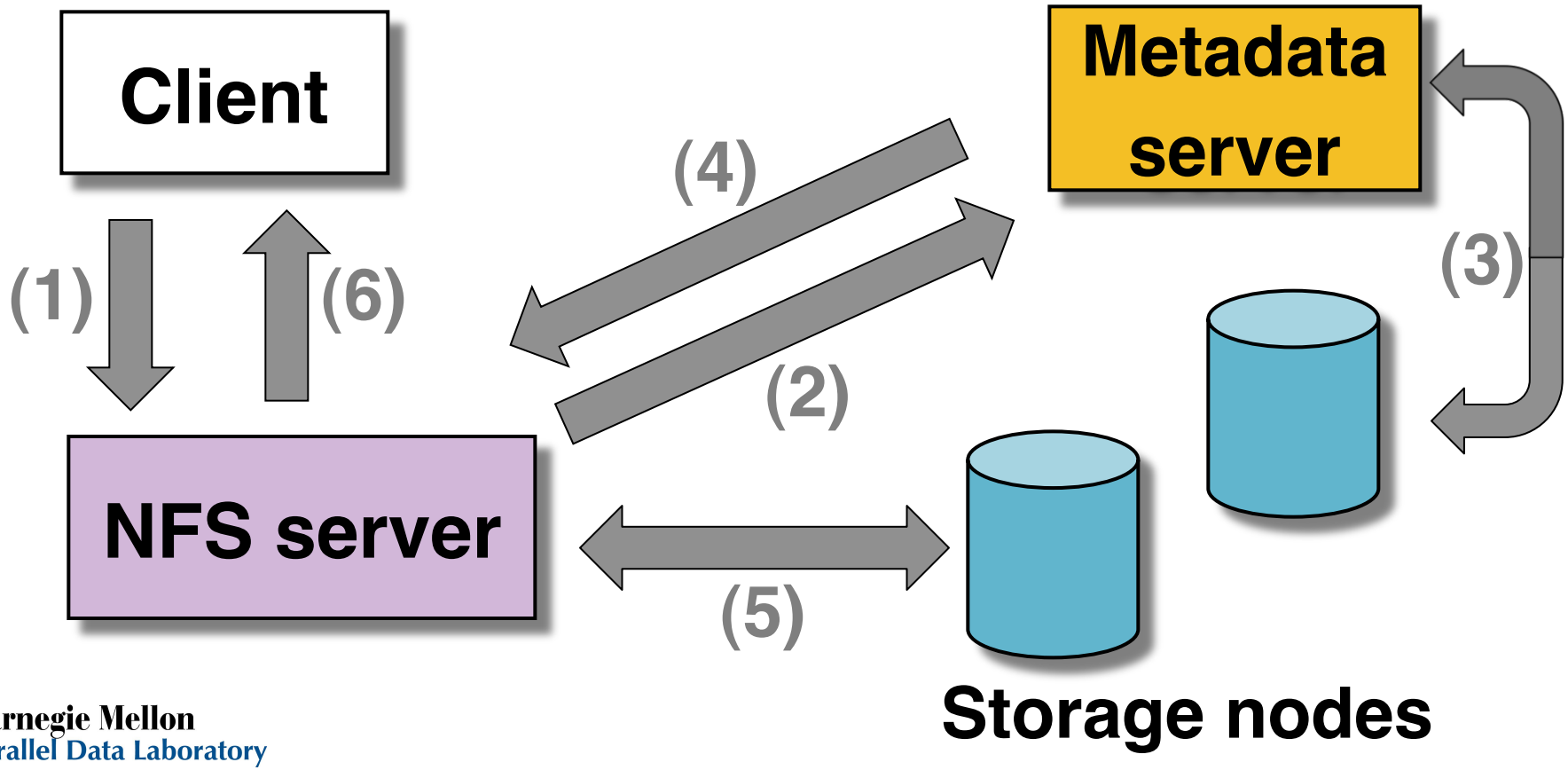
- **Request-flow comparison**
  - Helps localize performance changes
  - Key insight: Changes manifest as mutations in request timing/structure

Perf. debugging a feature addition



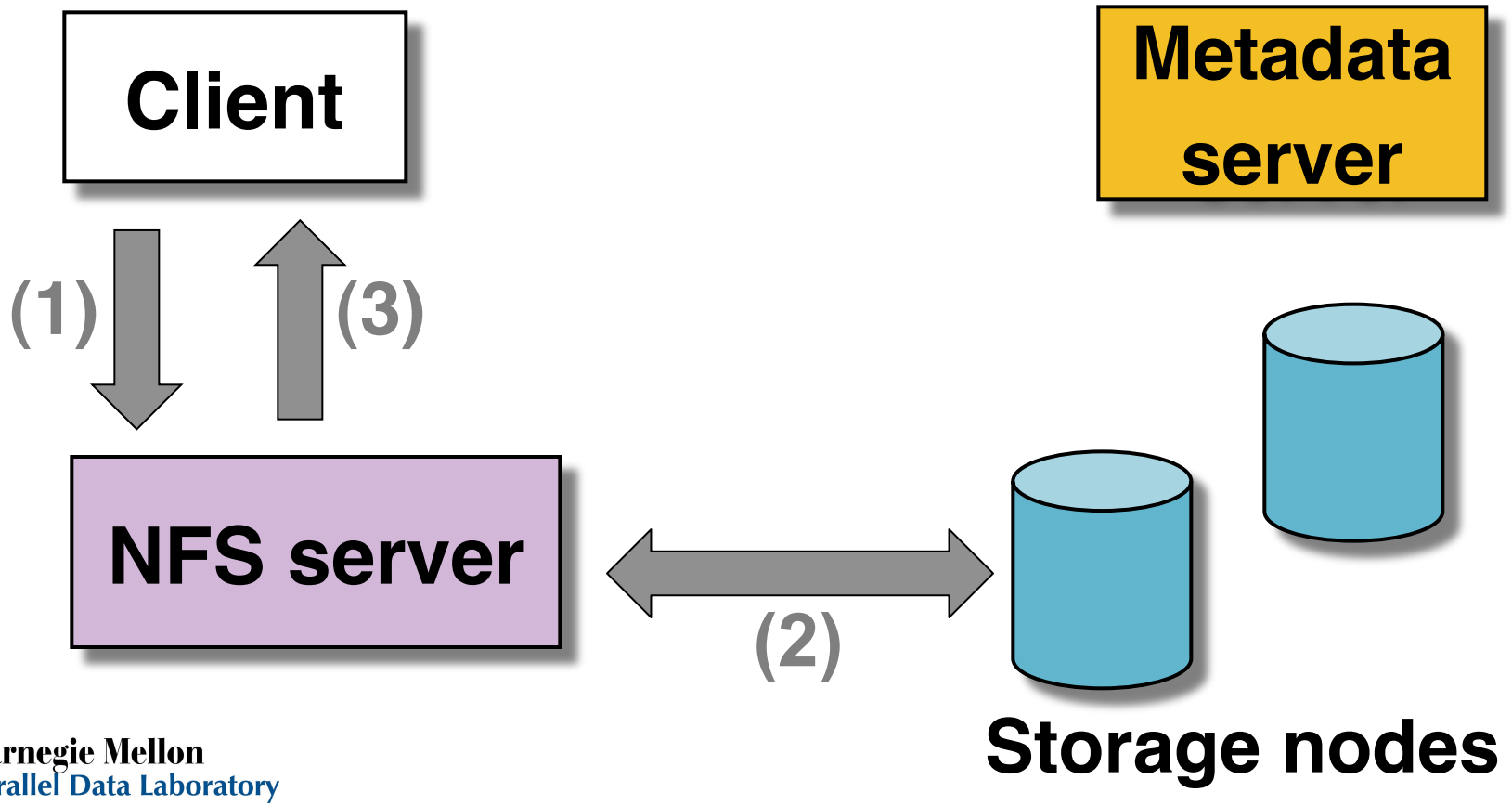
Perf. debugging a feature addition

- Before addition:
  - Every file access needs a MDS access



Perf. debugging a feature addition

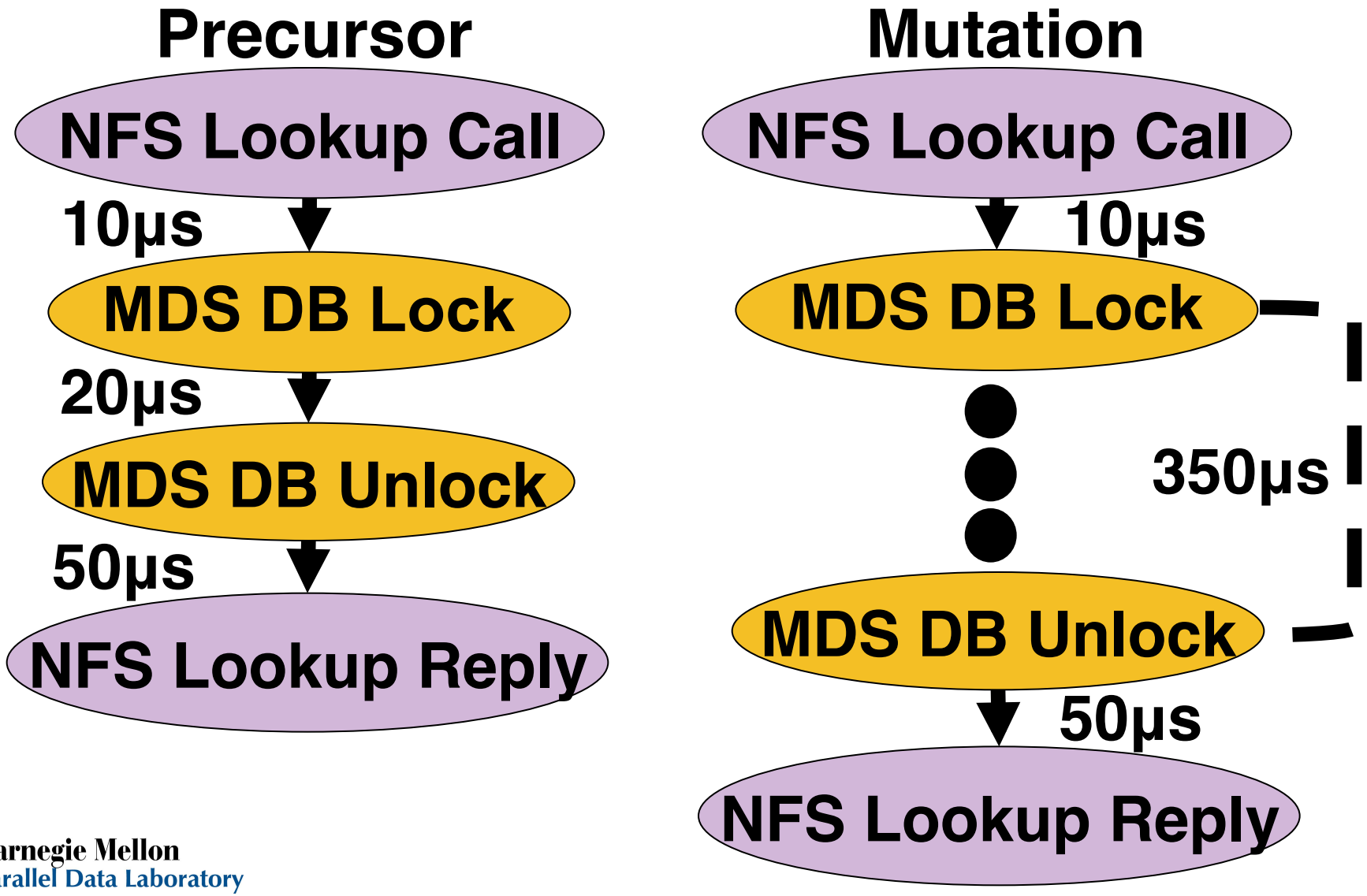
- After: Metadata prefetched to clients
  - Most requests **don't** need MDS access



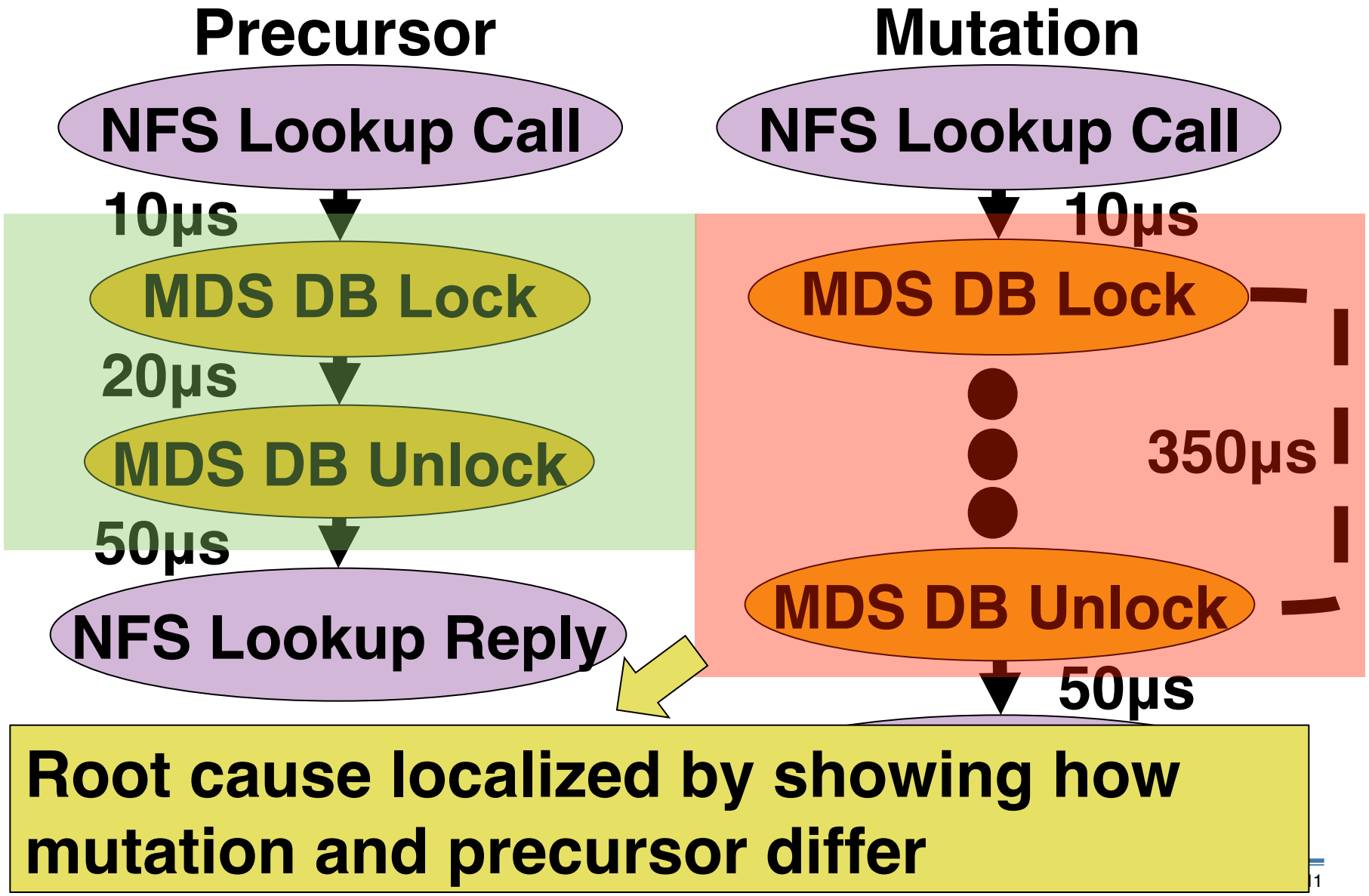
Perf. debugging a feature addition

- Adding metadata prefetching reduced performance instead of improving it (!)
- How to efficiently diagnose this?

Request-flow comparison will show



Request-flow comparison will show



Request-flow comparison

- Identifies distribution changes
  - Distinct from anomaly detection
    - E.g., Magpie, Pinpoint, etc.

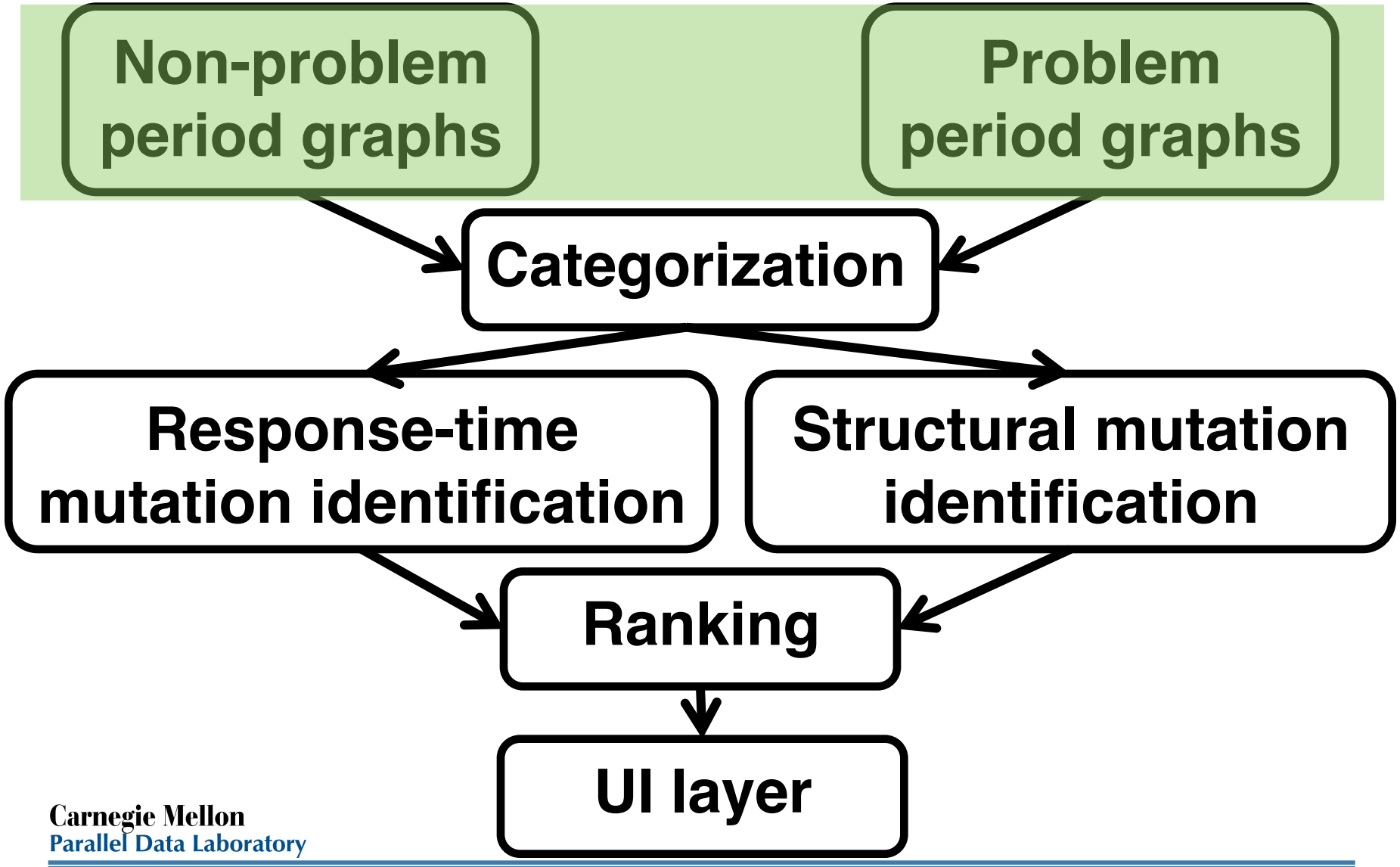
- Satisfies many use cases
  - Performance regressions/degradations
  - Eliminating the system as the culprit

Contributions

- Heuristics for identifying mutations, precursors, and for ranking them
  - Implementation in Spectroscope

- Use of Spectroscope to diagnose
  - Unsolved problems in Ursa Minor
  - Problems in Google services

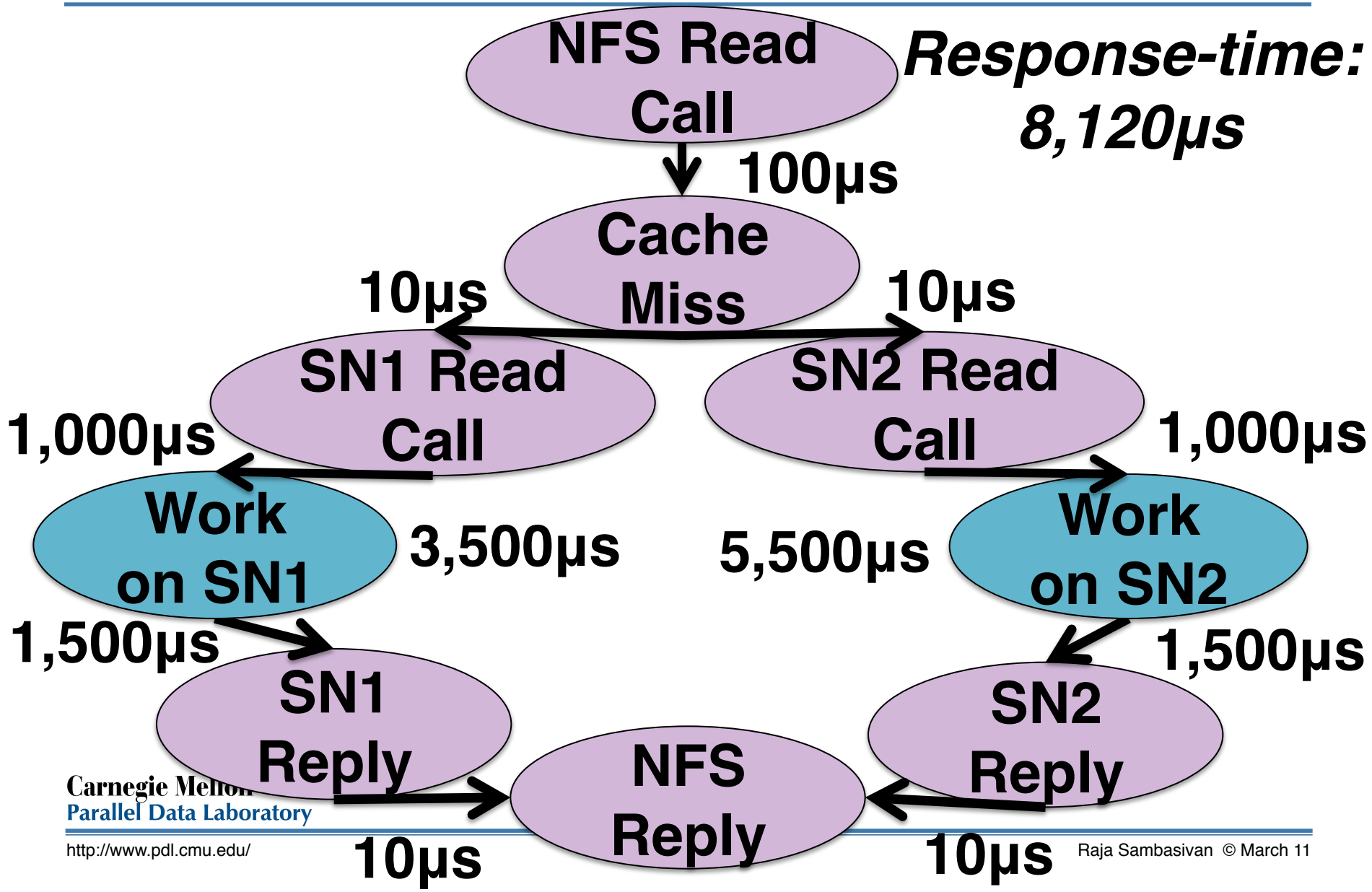
Spectroscope workflow



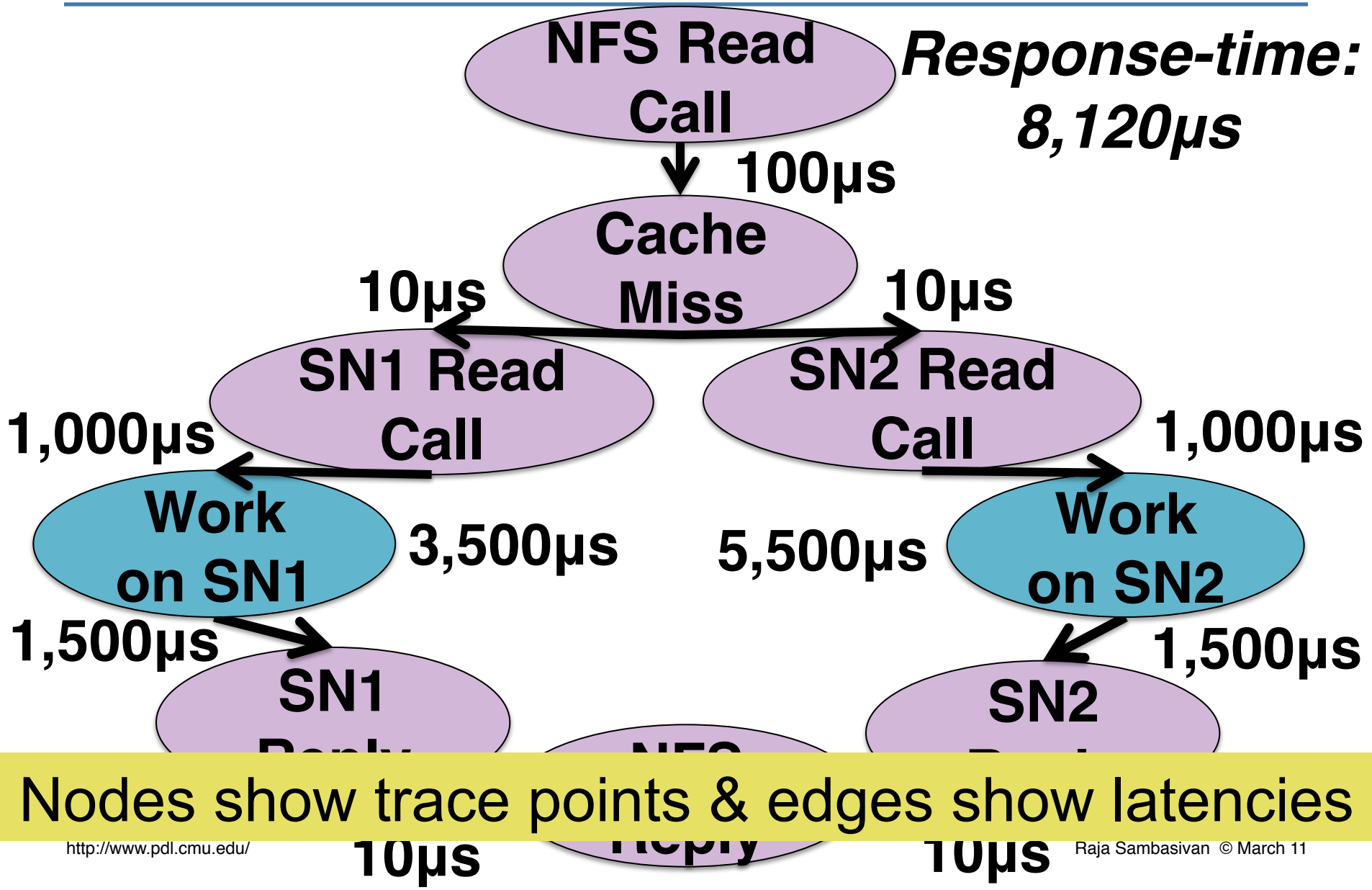
Graphs via end-to-end tracing

- Used in research & production systems
  - E.g., Magpie, X-Trace, Google's Dapper
- Works as follows:
  - Tracks trace points touched by requests
  - Request-flow graphs obtained by stitching together trace points accessed
- Yields < 1% overhead w/req. sampling

Example: Graph for a striped read

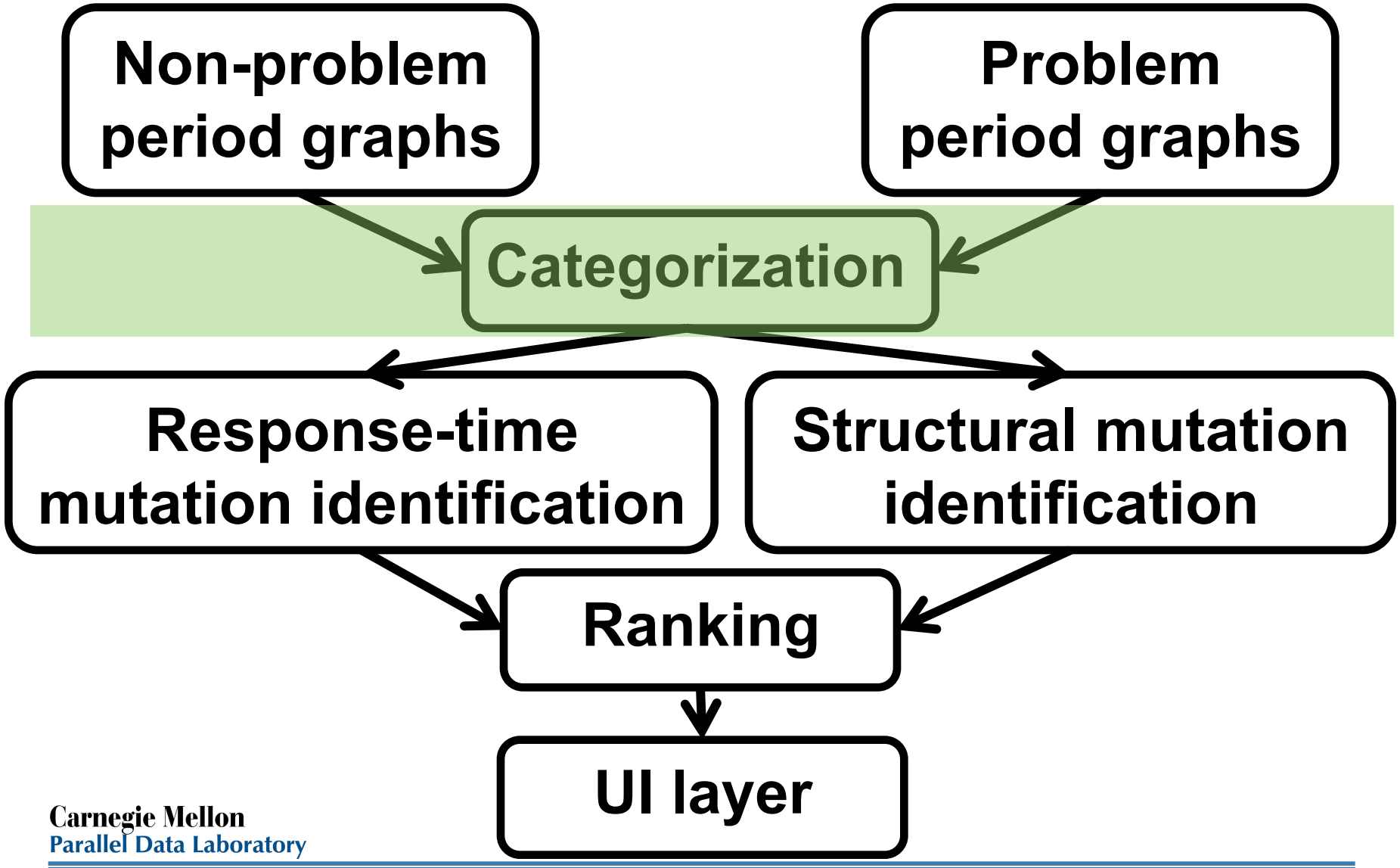


Example: Graph for a striped read



Nodes show trace points & edges show latencies

Spectroscope workflow



Categorization step

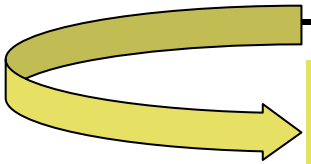
- Necessary since it is meaningless to compare individual requests flows

- Groups together similar request flows
  - Categories: basic unit for comparisons
  - Allows for mutation identification by comparing per-category distributions

Choosing what to bin into a category

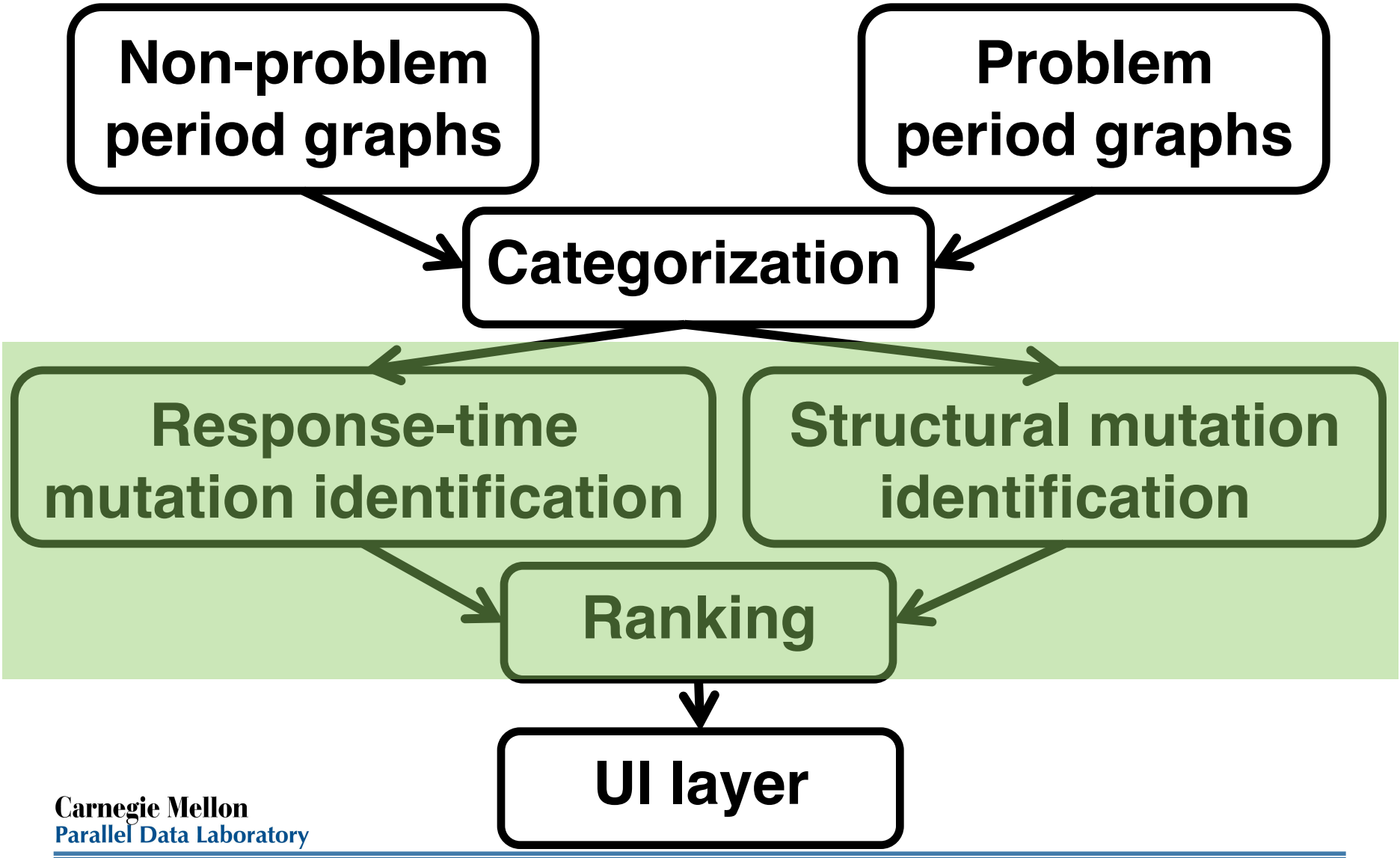
- Our choice: identically structured reqs
  - Uses same path/similar cost expectation

- Same path/similar costs notion is valid
  - For 88—99% of Ursa Minor categories
  - For 47—69% of Bigtable categories
    - –Lower value due to sparser trace points
    - –Lower value also due to contention



Aside: Categorization can be used to localize problematic sources of variance

Spectroscope workflow



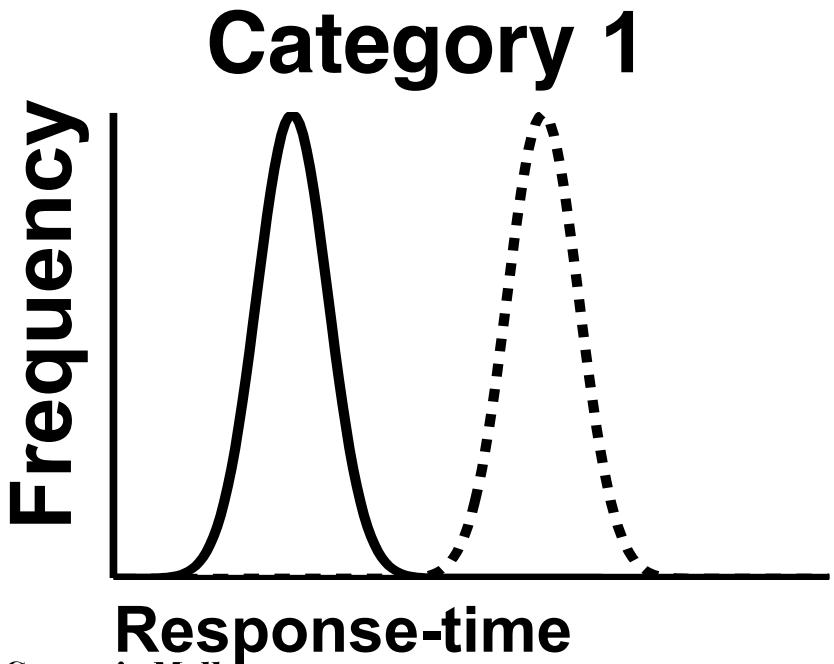
Type 1: Response-time mutations

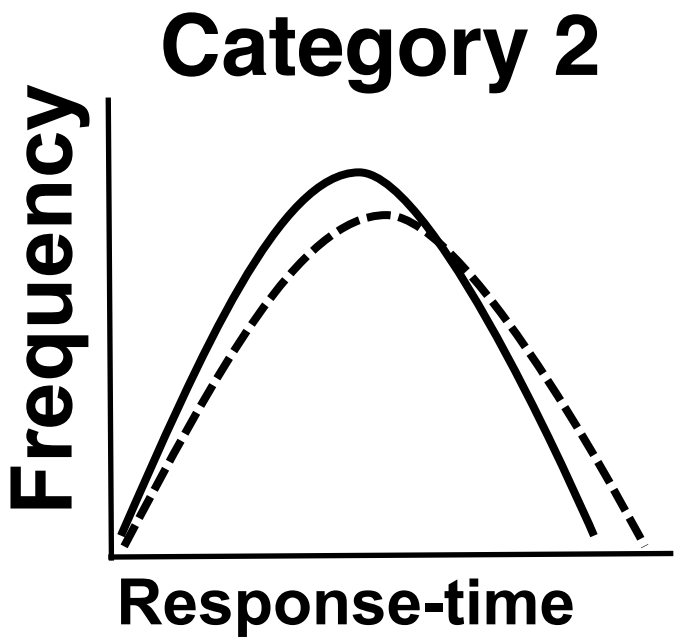
- Requests that:
  - are structurally identical in both periods
  - have larger problem period latencies

- Root cause localized by…
  - identifying interactions responsible

Categories w/response-time mutations

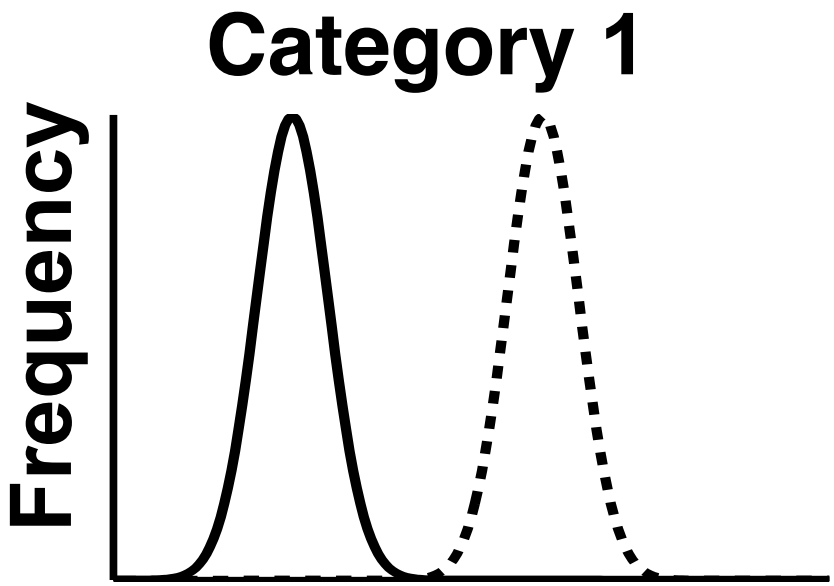
- Identified via use of a hypothesis test
  - Sets apart natural variance from mutations
  - Also used to find interactions responsible

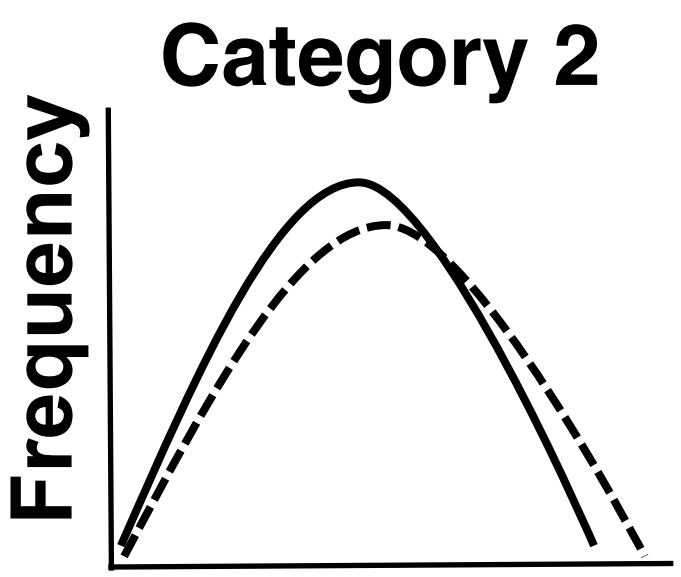




Categories w/response-time mutations

- Identified via use of a hypothesis test
  - Sets apart natural variance from mutations
  - Also used to find interactions responsible





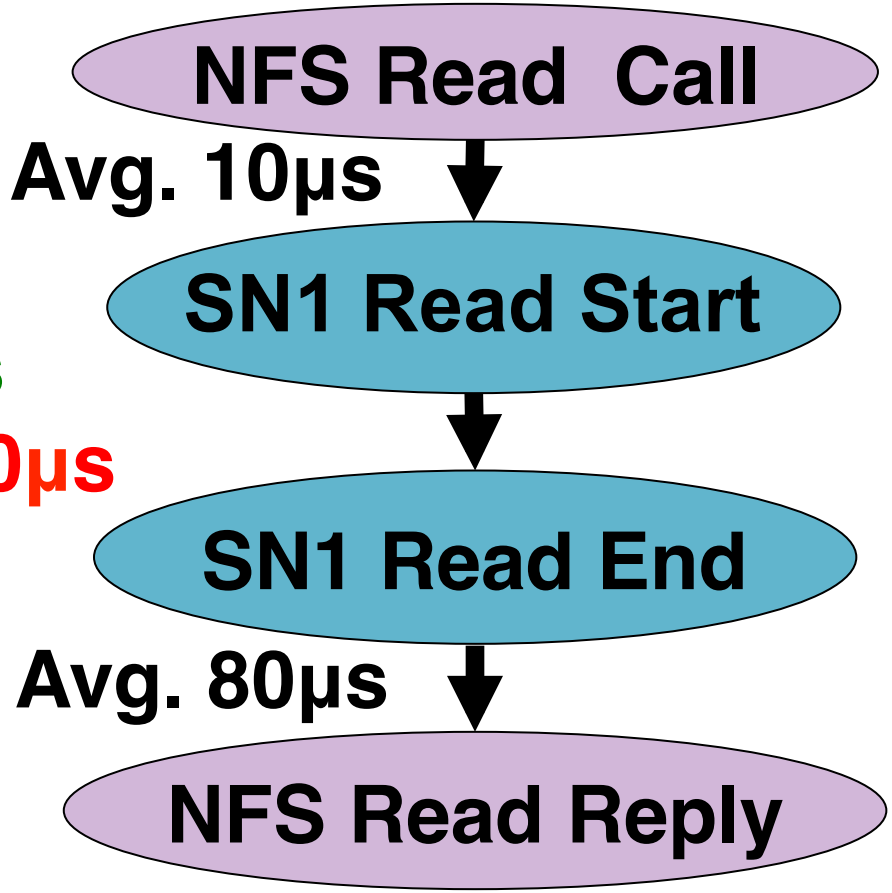
ID'd as containing mutations

Not ID'd as containing mutations

Response-time mutation example

Avg. non-problem response time: 110µs

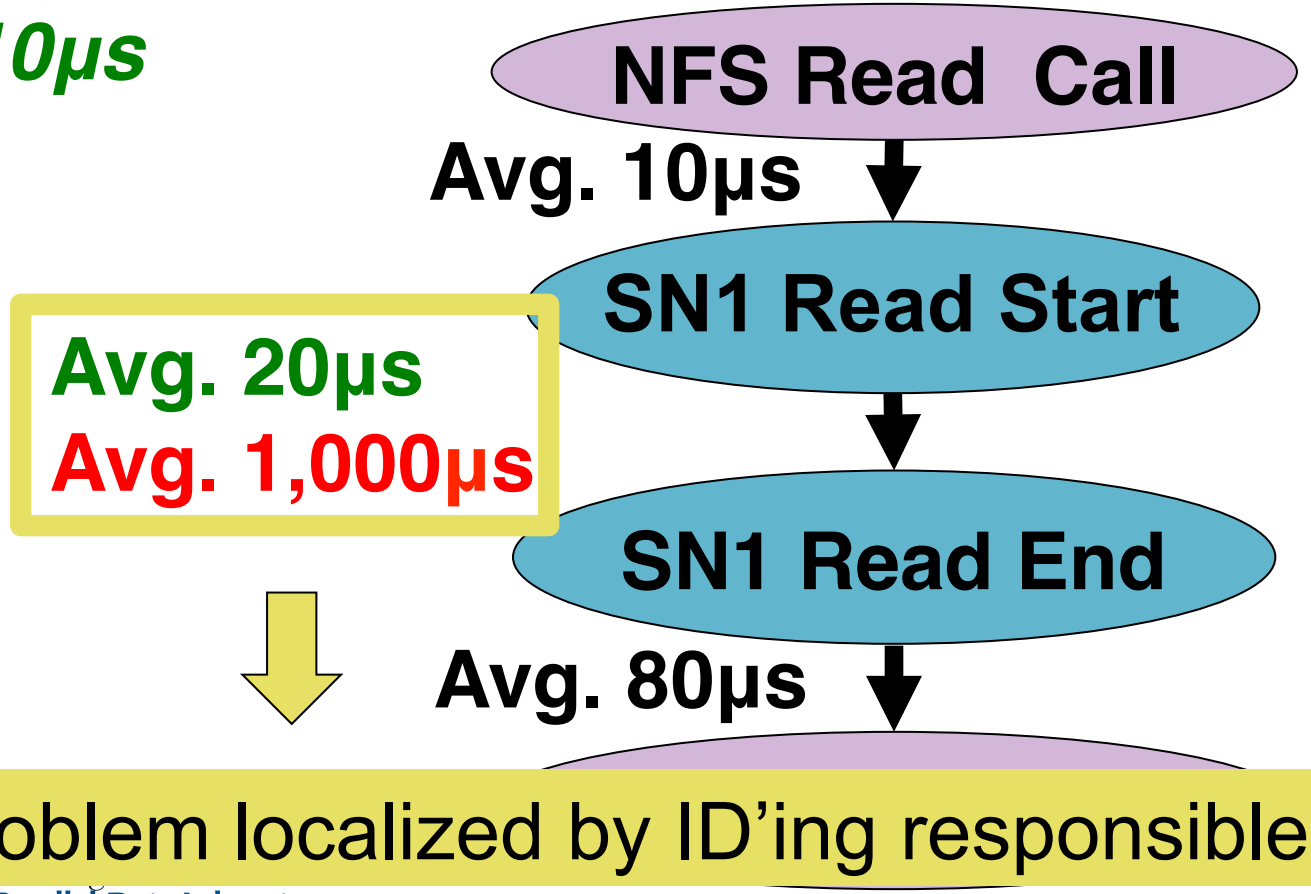
Avg. problem response time: 1,090µs



Response-time mutation example

Avg. non-problem response time: 110µs

Avg. problem response time: 1,090µs



Problem localized by ID'ing responsible interaction

Type 2: Structural mutations

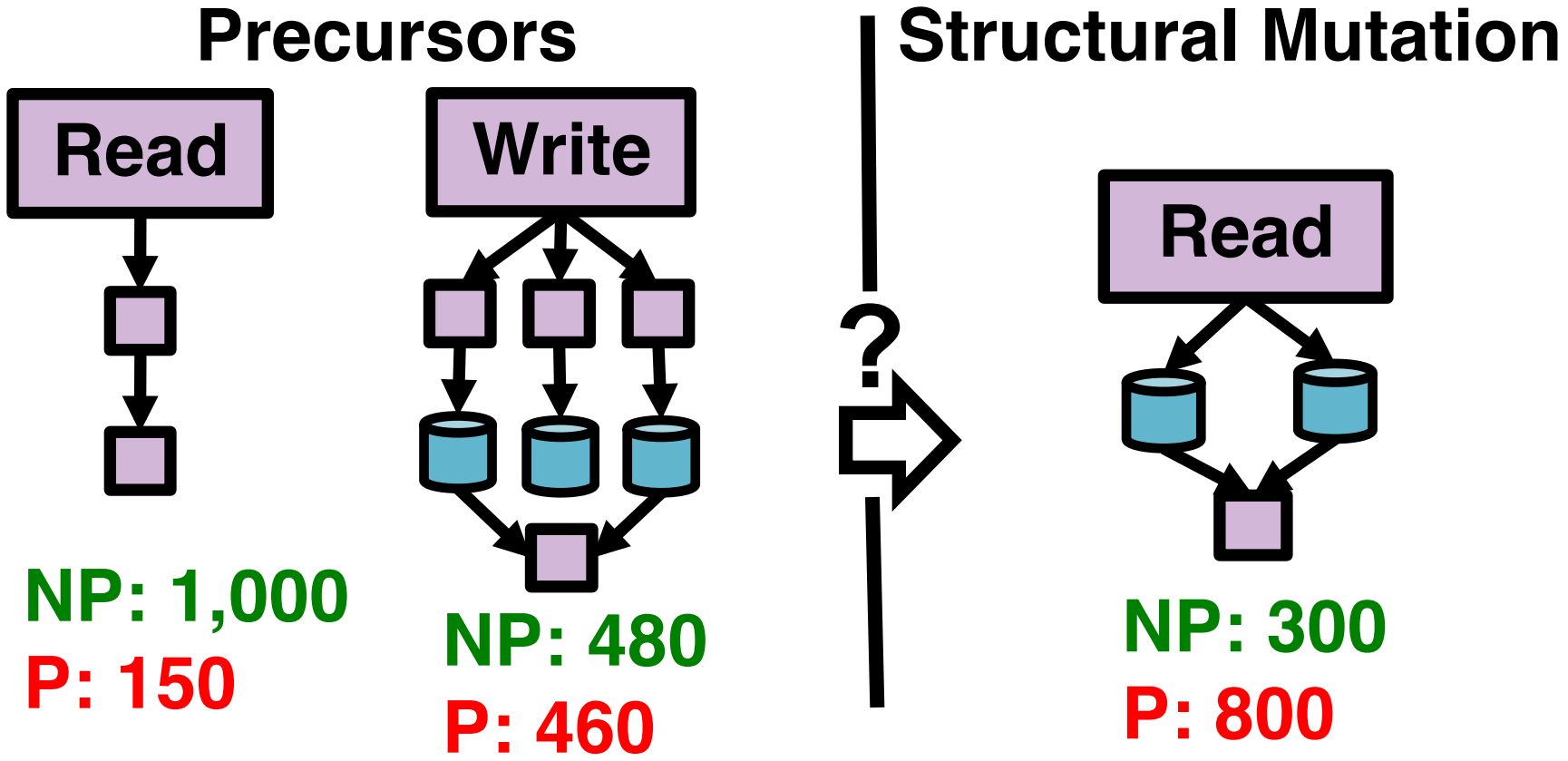
- Requests that:
  - take different paths in the problem period
- Root caused localized by…
  - identifying their precursors
    - Likely path during the non-problem period
  - id'ing how mutation & precursor differ

ID'ing categories w/structural mutations

- Assume similar workloads executed
  - Categories with more problem period requests contain mutations
  - Reverse true for precursor categories

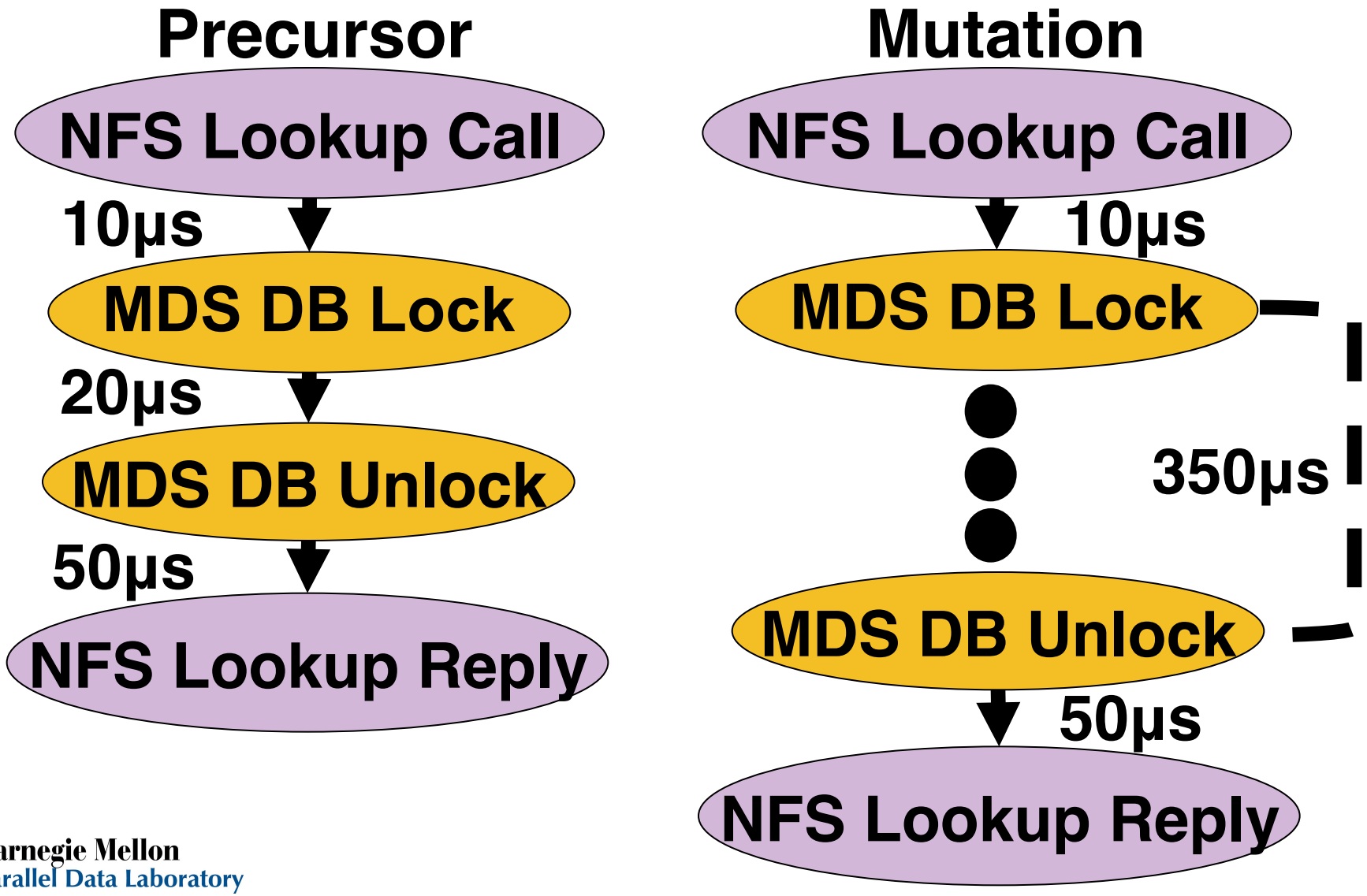
- Threshold used to differentiate natural variance from categories w/mutations

Mapping mutations to precursors

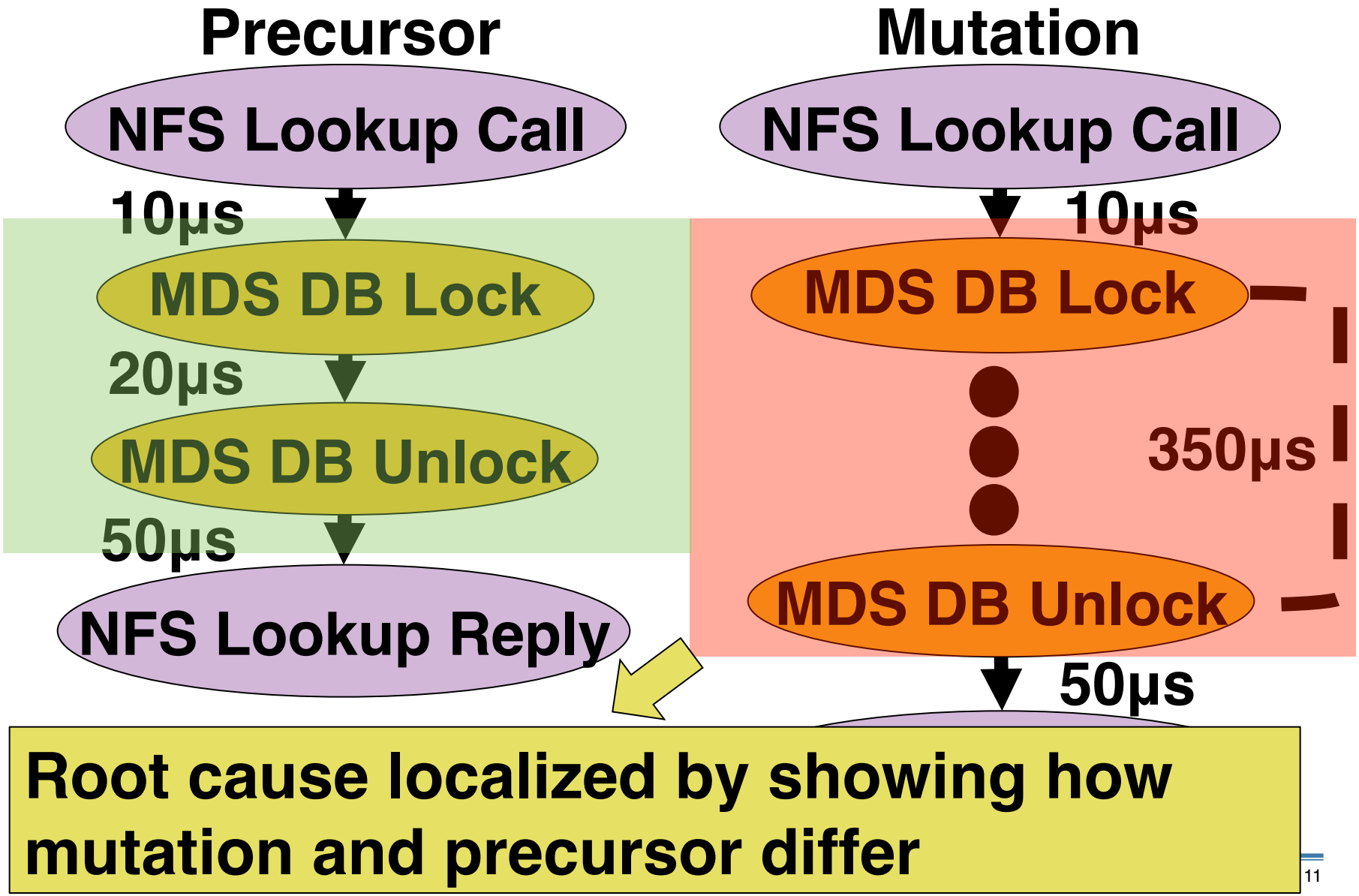


- Accomplished using three heuristics
  - See paper for details

Example structural mutation



Example structural mutation



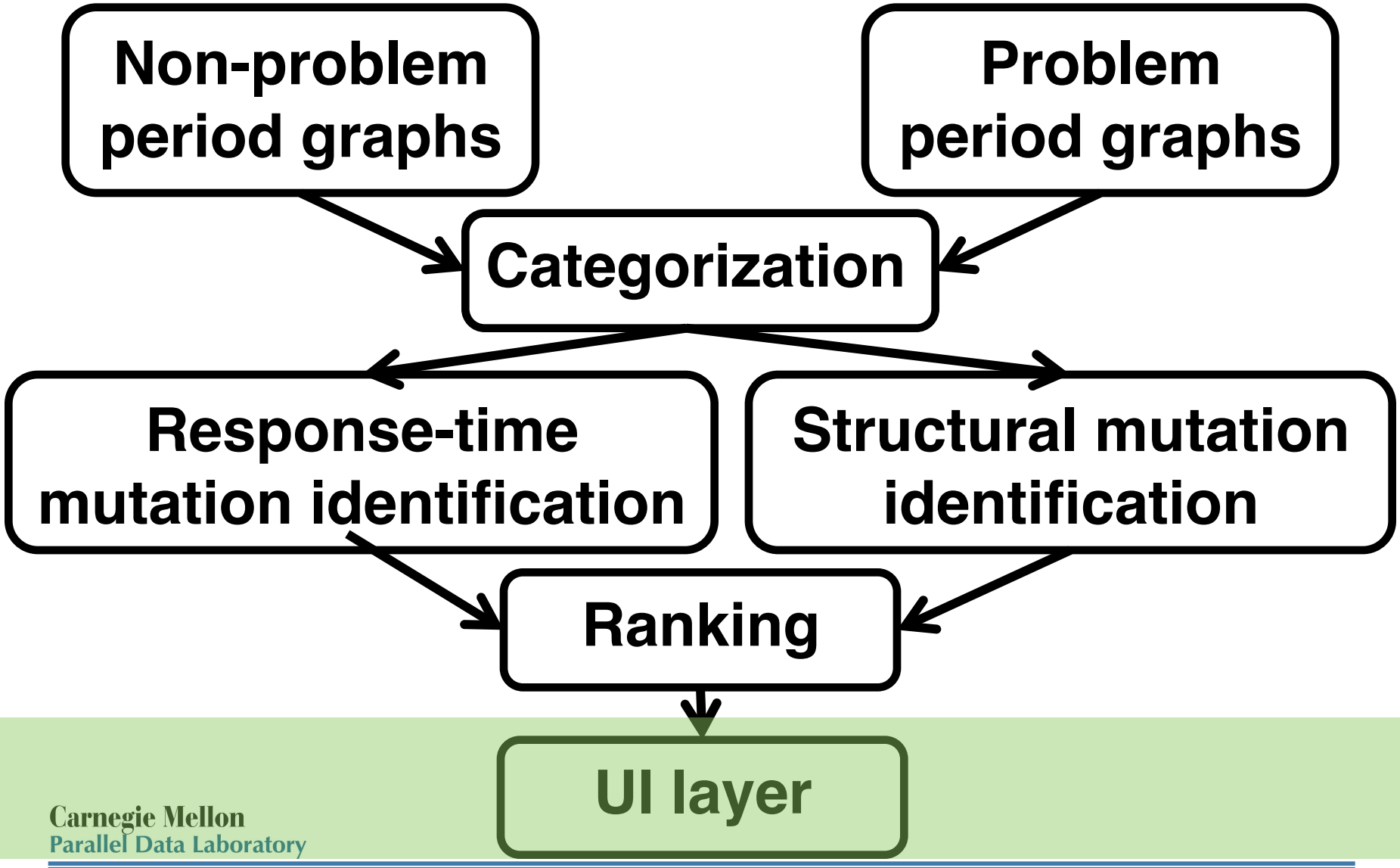
Root cause localized by showing how mutation and precursor differ

Ranking of categories w/mutations

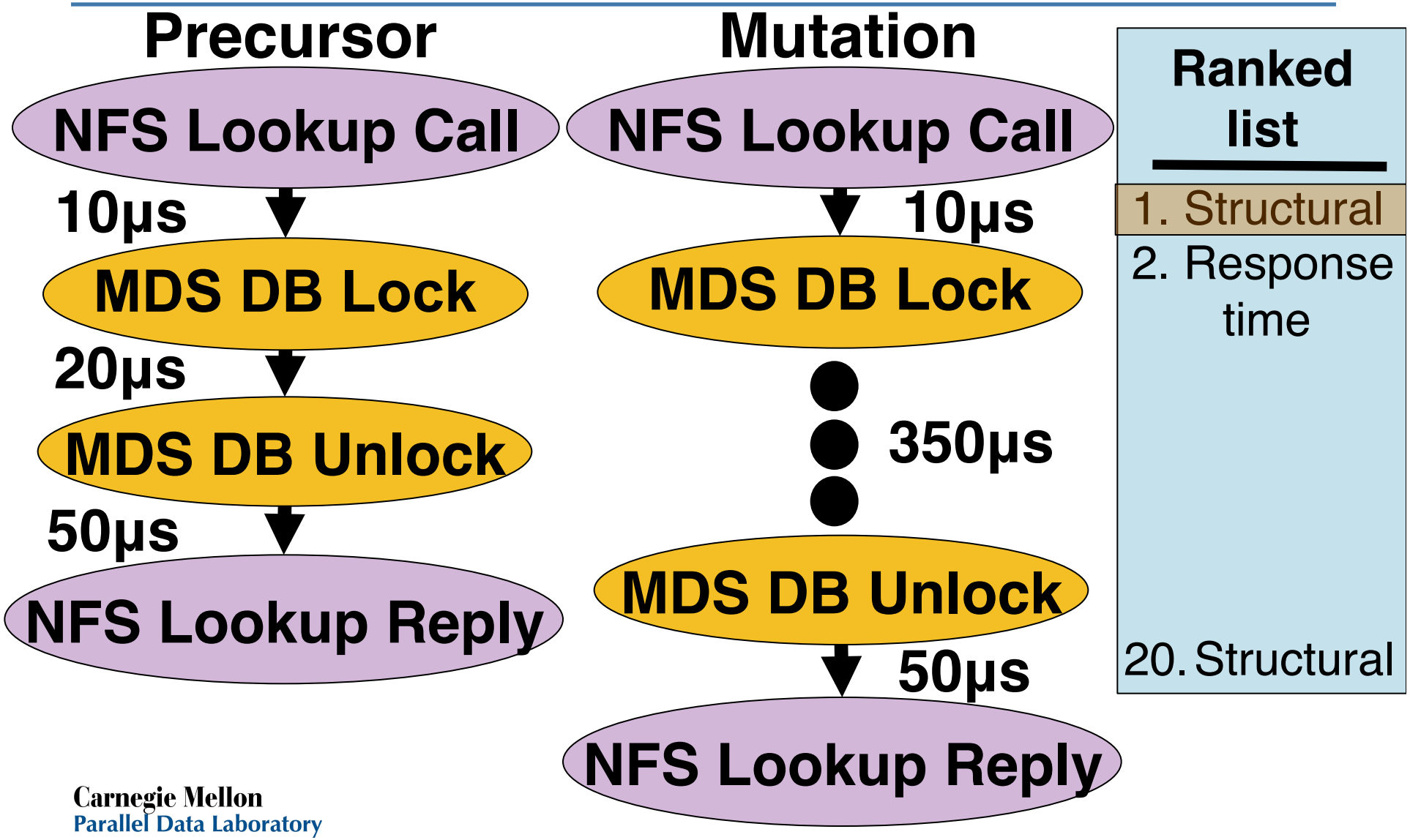
- Necessary for two reasons
  - There may be more than one problem
  - One problem may yield many mutations

- Rank based on:
  - # of reqs affected * Δ in response time

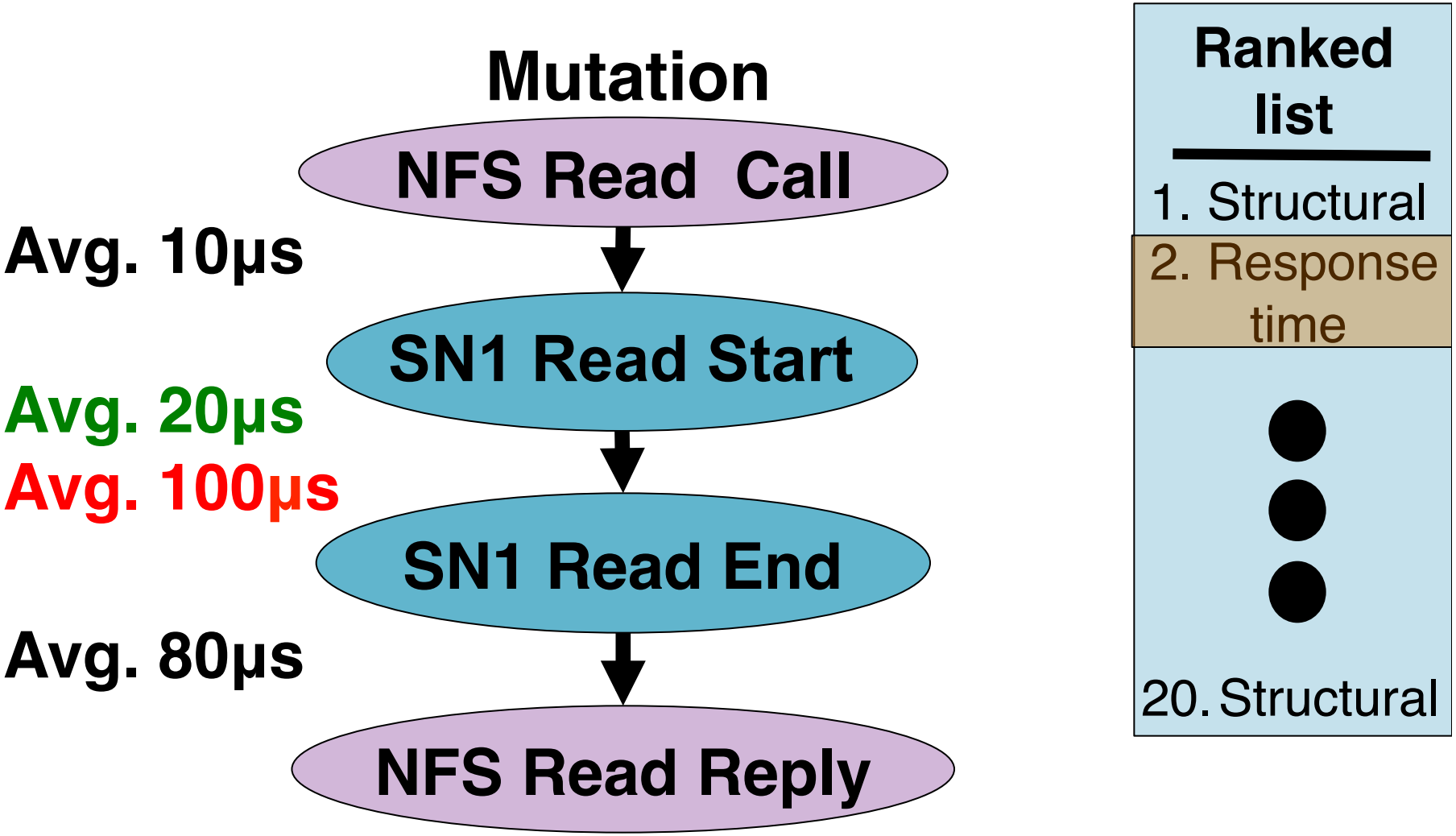
Spectroscope workflow



Putting it all together: The UI



Putting it all together: The UI



Outline

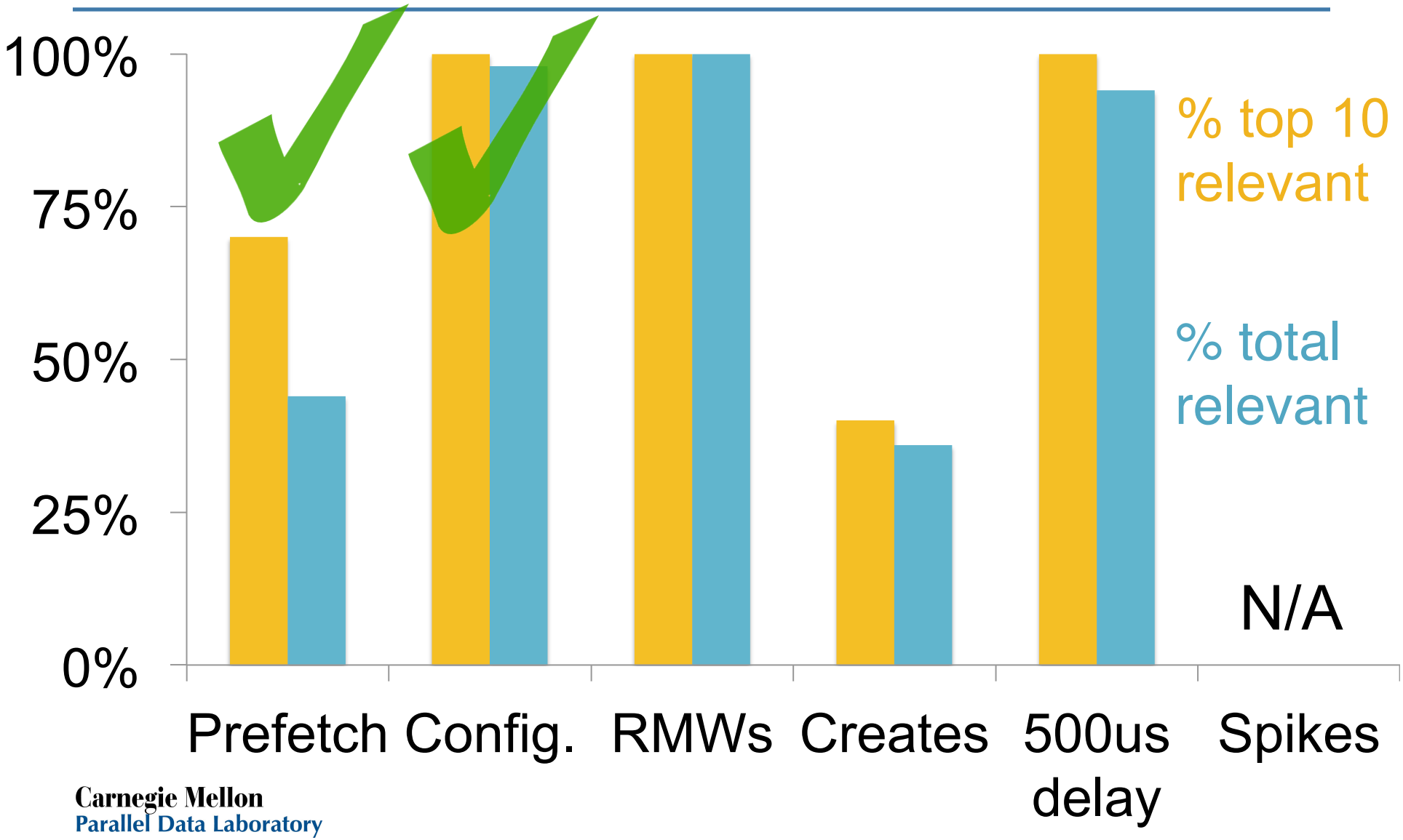
- Introduction
- End-to-end tracing & Spectroscope
- Ursa Minor & Google case studies
- Summary

Ursa Minor case studies

- Used Spectroscope to diagnose ***real*** performance problems in Ursa Minor
  - Four were previously unsolved

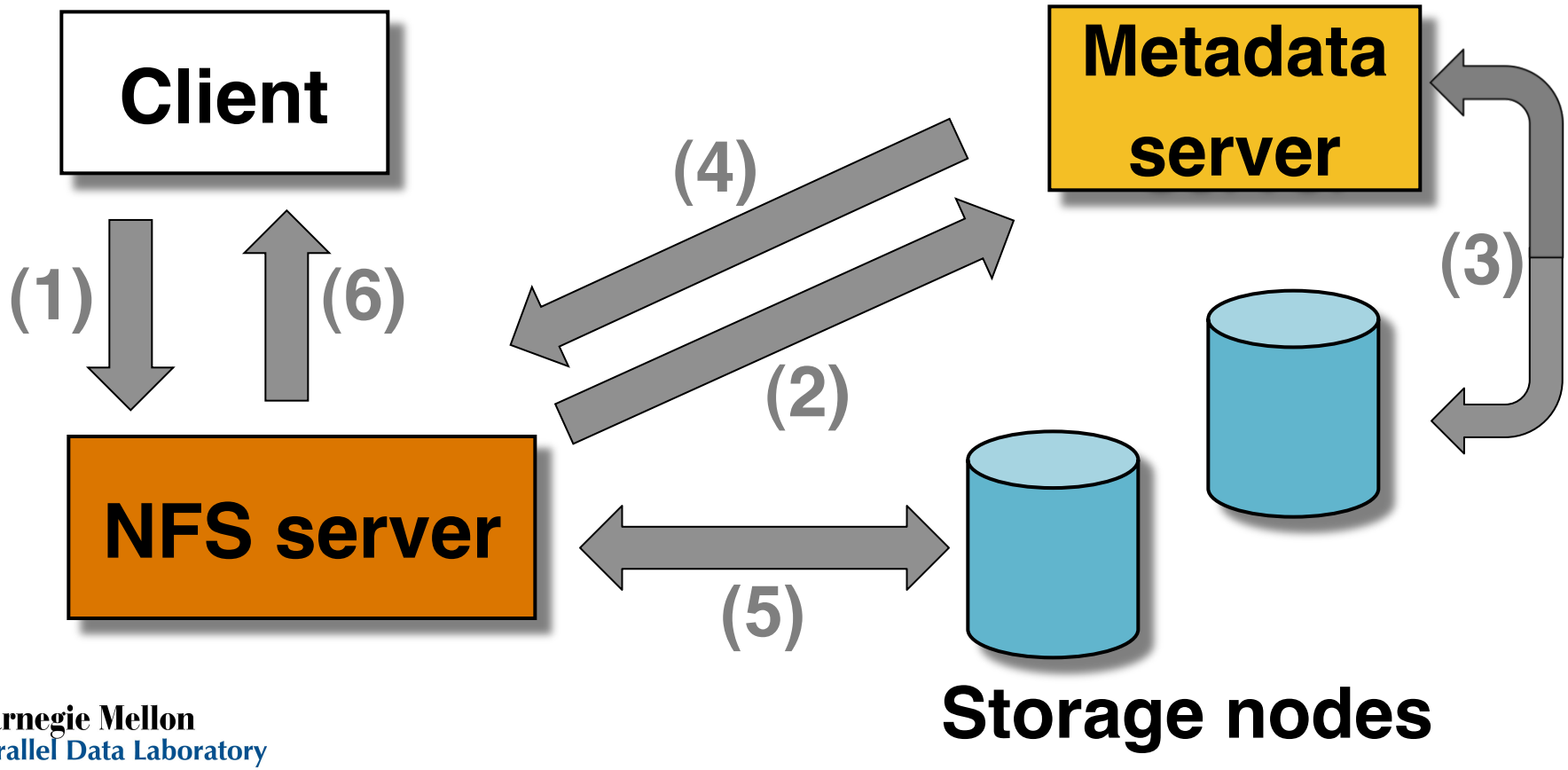
- Evaluated ranked list by measuring
  - % of top 10 results that were relevant
  - % of total results that were relevant

Quantitative results



Ursa Minor 5-component config

- All case studies use this configuration



Case 1: MDS configuration problem

- **Problem**: Slowdown in key benchmarks seen after large code merge

Comparing request flows

- Identified 128 mutation categories:
  - Most contained structural mutations
  - Mutations and precursors differed only in the storage node accessed

- Localized bug to unexpected interaction between MDS & the data storage node
  - But, unclear why this was happening

Further localization

- Thought mutations may be caused by changes in low-level parameters
  - E.g., function call parameters, etc.

- Identified parameters that separated 1st-ranked mutation from its precursors
  - Showed changes in encoding params
  - Localized root cause to two config files

Root cause: Change to an obscure config file

Inter-cluster perf. at Google

- Load tests run on same software in two datacenters yielded very different perf.
  - Developers said perf. should be similar

Comparing request flows

- Revealed many mutations
  - High-ranked ones were response-time

- Responsible interactions found both within the service and in dependencies
  - Led us to suspect root cause was issue with the slower load test's datacenter

Root cause: Shared Bigtable in slower load test's datacenter was not working properly

Summary

- Introduced *request-flow comparison* as a new way to diagnose perf. changes
- Presented algorithms for localizing problems by identifying mutations
- Showed utility of our approach by using it to diagnose real, unsolved problems

Related work (I)

[Abd-El-Malek05b]: **Ursa Minor: versatile cluster-based storage**. Michael Abd-El-Malek, William V. Courtright II, Chuck Cranor, Gregory R. Ganger, James Hendricks, Andrew J. Klosterman, Michael Mesnier, Manish Prasad, Brandon Salmon, Raja R. Sambasivan, Shafeeq Sinnamohideen, John D. Strunk, Eno Thereska, Matthew Wachs, Jay J. Wylie. FAST, 2005.

[Barham04]: **Using Magpie for request extraction and workload modelling**. Paul Barham, Austin Donnelly, Rebecca Isaacs, Richard Mortier. OSDI, 2004.

[Chen04]: **Path-based failure and evolution management**. Mike Y. Chen, Anthony Accardi, Emre Kiciman, Jim Lloyd, Dave Patterson, Armando Fox, Eric Brewer. NSDI, 2004.

Related work (II)

[Cohen05]: **Capturing, indexing, and retrieving system history**. Ira Cohen, Steve Zhang, Moises Goldszmidt, Terrence Kelly, Armando Fox. SOSP, 2005.

[Fonseca07]: **X-Trace: A Pervasive Network Tracing Framework**. Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker, Ion Stoica. NSDI, 2007.

[Hendricks06]: **Improving small file performance in object-based storage**. James Hendricks, Raja R. Sambasivan, Shafeeq Sinnamohideen, Gregory R. Ganger. Technical Report CMU-PDL-06-104, 2006.

Related work (III)

[Reynolds06]: **Detecting the unexpected in distributed systems**. Patrick Reynolds, Charles Killian, Janet L. Wiener, Jeffrey C. Mogul, Mehul A. Shah, Amin Vahdat. NSDI, 2006.

[Sambasivan07]: **Categorizing and differencing system behaviours**. Raja R. Sambasivan, Alice X. Zheng, Eno Thereska. HotAC II, 2007.

[Sambasivan10]: **Diagnosing performance problems by visualizing and comparing system behaviours**. Raja R. Sambasivan, Alice X. Zheng, Elie Krevat, Spencer Whitman, Michael Stroucken, William Wang, Lianghong Xu, Gregory R. Ganger. CMU-PDL-10-103.

Related work (IV)

[Sambasivan11]: **Automation without predictability is a recipe for failure**. Raja R. Sambasivan and Gregory R. Ganger. Technical Report CMU-PDL-11-101, 2011.

[Sigelman10]: **Dapper, a large-scale distributed tracing infrastructure.** Benjamin H. Sigelman, Luiz Andre´ Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, Chandan Shanbhag. Technical report 2010-1, 2008.

[Thereska06b]: **Stardust: tracking activity in a distributed storage system**. Eno Thereska, Brandon Salmon, John Strunk, Matthew Wachs, Michael Abd-El-Malek, Julio Lopez, Gregory R. Ganger. SIGMETRICS, 2006.