Lawrence Livermore National Laboratory

Resilient and Efficient High-Performance Computing via Application Behavior Analysis

Greg Bronevetsky

Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DEAC52-07NA27344 and supported by the Department of Energy Early Career Research Program.

Good Old Days

Plentiful Resources

Optimize and Done

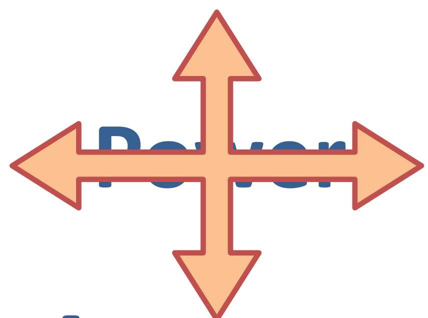
Future

Severe Resource Constraints

Future

Severe Resource Constraints

Power/Energy

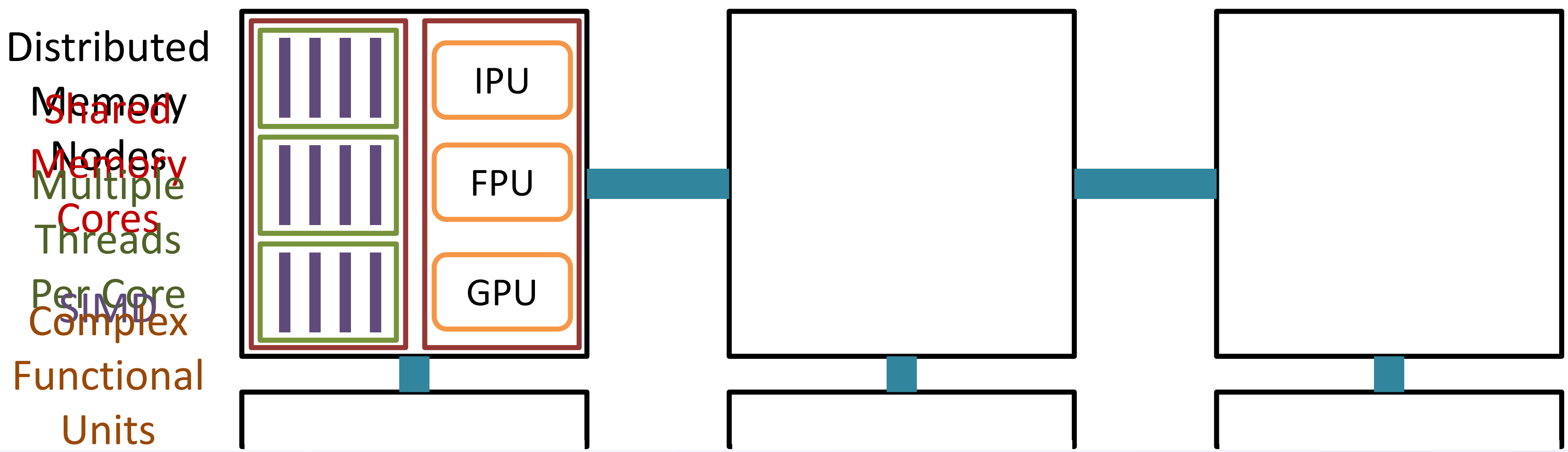
Performance  Usability

Accuracy

Many-Dimensional <u>Productivity</u> Optimization

Optimization solutions induce complexity in application and system design/behavior

- Systems Complexity
  - Failures (soft errors, fail-stop crashes)
  - Static and dynamic performance variation
    (dynamic voltage scaling, variable guardbands)
  - Complex, heterogeneous hierarchies



Optimization solutions induce complexity in application and system design/behavior

- **Systems Complexity**
  - Failures (soft errors, fail-stop crashes)
  - Static and dynamic performance variation
    (dynamic voltage scaling, variable guardbands)
  - Complex, heterogeneous hierarchies
- **Application Complexity**
  - Adaptation to problem structure
    (Adaptive refinement, sparse systems)
  - Dynamic load-balancing
  - Coupled multi-physics simulations

Key Challenge: co-managing applications and system to achieve high productivity

- Application and system flexibility induces very large interaction space:
  Challenge and Opportunity

Weather Simulation

– 1 ... sor

– ...

When is high precision needed?
- High wind speed?
- High temperature?
- …?

– Can vary precision over space/time regions

System

– La... essors

– P... m

– R...

s... e

When should system reduce frequency?
- High temperature?
- Erratic timing?
- …?

– Can reduce frequency if problems arise

Challenging to control within each component

Key Challenge: co-managing applications and system to achieve high productivity

- Application and system flexibility induces very large interaction space: **Challenge and Opportunity**

<u>Weather Simulation</u>

When is high precision needed?
- High wind speed?
- High temperature?
- …?

<u>System</u>

When should system reduce frequency?
- High temperature?
- Erratic timing?
- …?

Cross-component optimization is more productive and challenging

Need techniques that productively harness application and system flexibility

- Combination of individual components has very complex interactions
- Productive control requires analysis of behavior and intelligence to guide it
  - Migrate load based both on simulated wind speeds and processor temperature
  - Reduce voltage to save power while meeting accuracy bound in face of hardware errors
- Minimal changes to existing source code

Detailed **Analysis** of Application and System

↓

Informed **Action** Based on Analysis

Analysis

- Dynamic statistical modeling
- Sensitive to properties of input

Meas

- Applicatio system me
- Quantify a

nent

- and
- applications
- use new

- Two end-to-end use-cases that demonstate utility of approach in sparse linear algebra
  - Resilience
  - Performance
- Highlights of deeper ongoing research to translate approach to real applications and systems

- Guide developers, system administrators

Use-case:
Error detection in sparse linear algebra

- Problem: radiation and voltage variation cause random corruptions of computations
- Checking matrix-vector multiplication $\boldsymbol{Ax}$

$$\boldsymbol{c}^{T}(\boldsymbol{Ax}) \; ?= \; (\boldsymbol{c}^{T}\boldsymbol{A})\boldsymbol{x}$$

Result

Match

Arbitrary

check vector

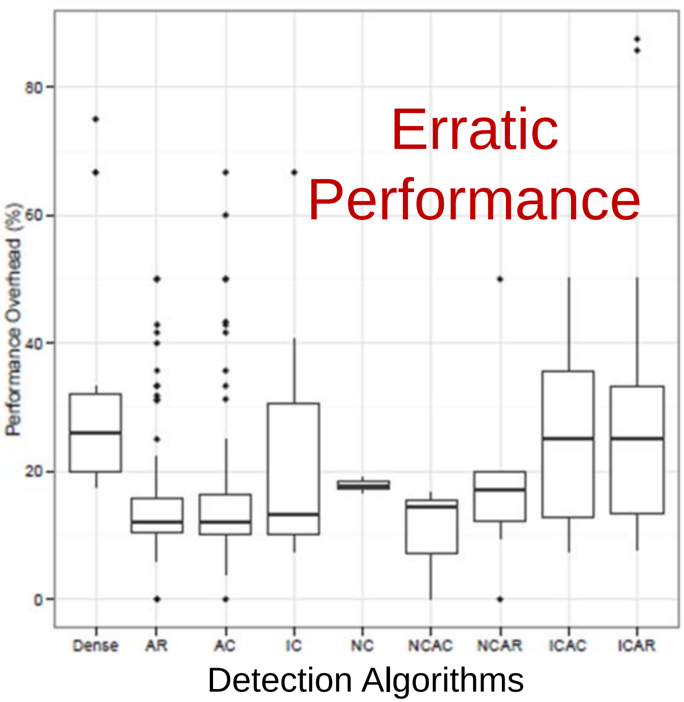
Computed offline

for each matrix A

- c can be vector of all 1s
- Or random choice of 90% 0s and 10% 1s
- Or Near A's null-space
- Or approximate solution to $\boldsymbol{c}^{T}\boldsymbol{A} = \boldsymbol{I}$

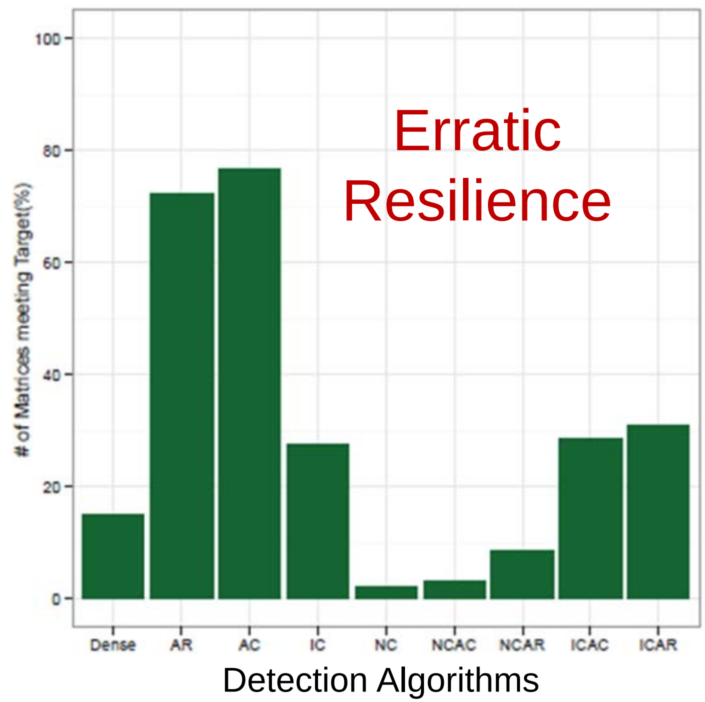
Performance and quality of error detectors varies highly across matrixes

Performance Overhead
of Error Detection



Good

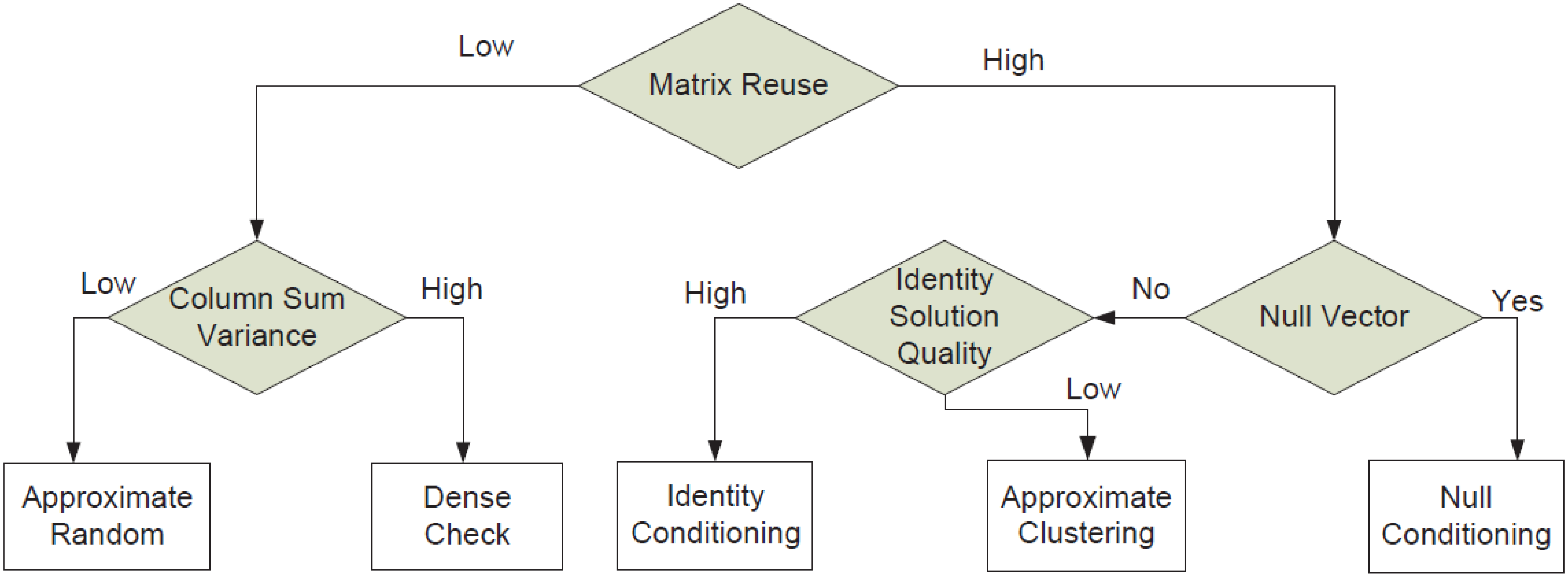
Accuracy of Error Detection
(Fraction of runs with high F-Score)



Good

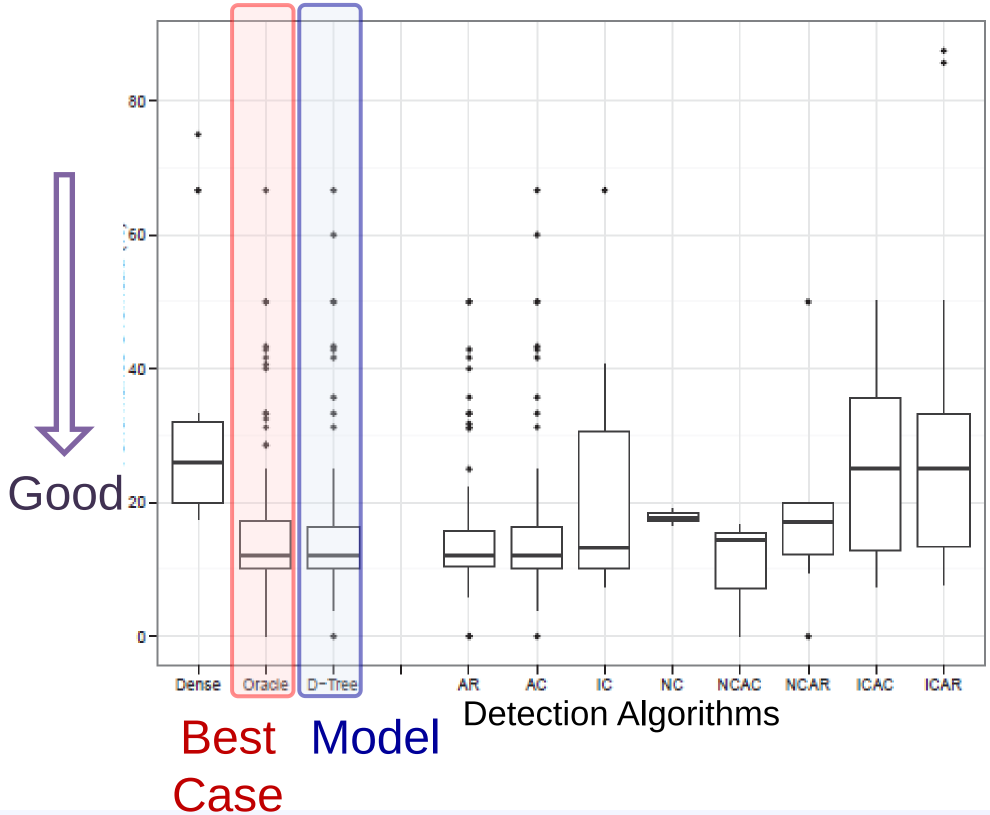
Measured over 100 sparse matrixes

Trained a matrix-sensitive statistical model of detector effectiveness



Model-guided error detector consistently efficient and accurate across input space

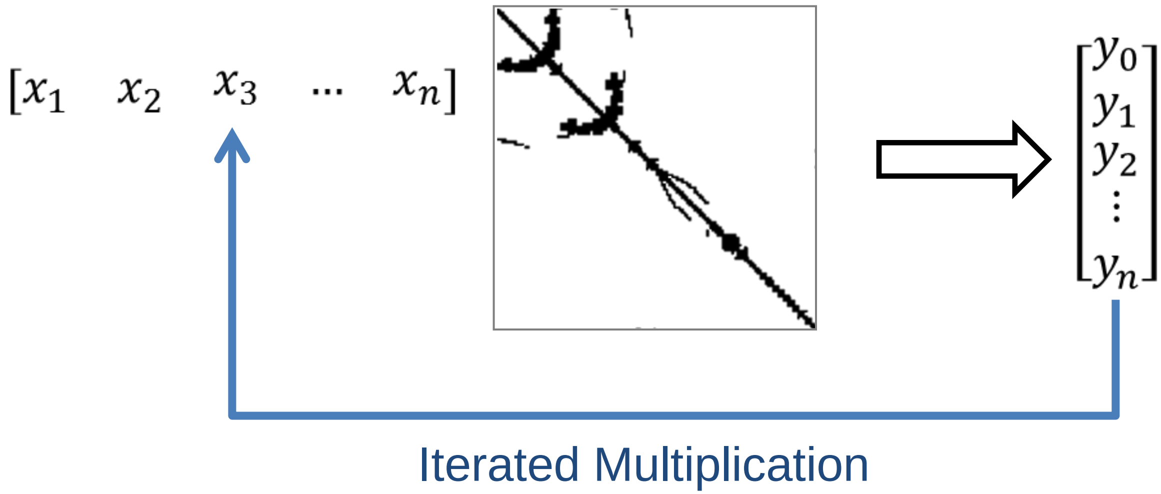
Performance Overhead of Error Detection



Accuracy of Error Detection
(Fraction of runs with high F-Score)

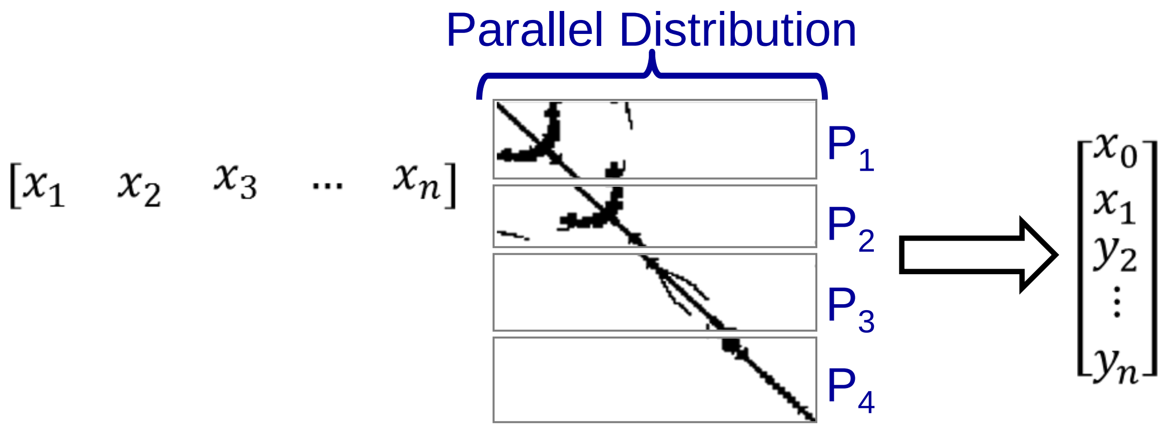
Use-case:
Optimization for sparse linear algebra

- Sparse iterative solvers repeatedly execute matrix-vector multiplication



Parallel performance depends on load balance and communication volume

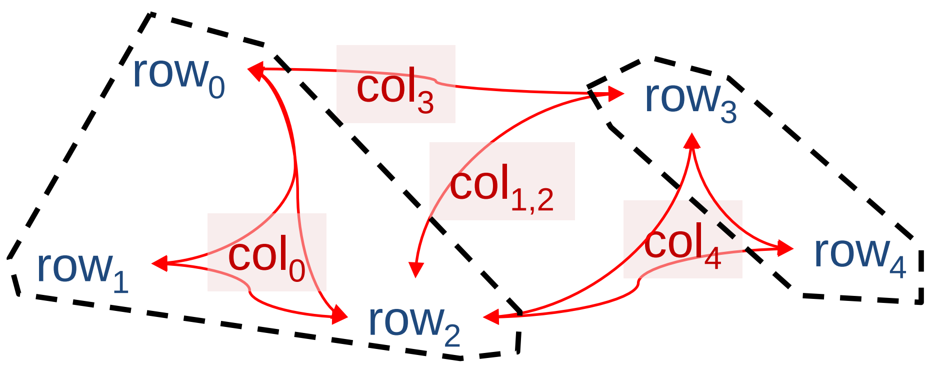
- Sparse iterative solvers repeatedly execute matrix-vector multiplication



- Rows distributed to
  - Balance computation
  - Reduce communication (values computed in iteration i sent to subset of row blocks for iteration i+1)

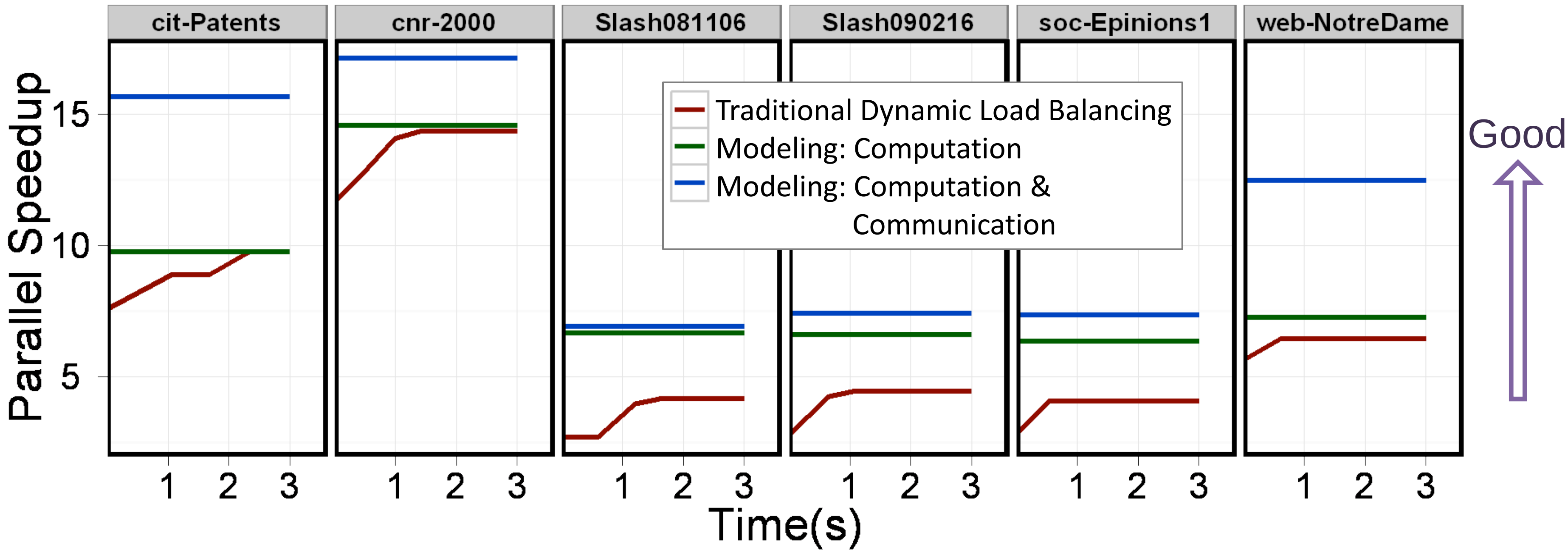
Computation and communication modeling simplifies data-dependent optimization

- **Computation model for each row**
  - Non-zero count (CPU use)
  - Distance from first to last column (Memory locality)
  - Statistically model row execution time
- **Communication model**
  - Columns shared by two rows



Partition graph to allocate load

Model-based scheduling improves performance and speeds scheduling



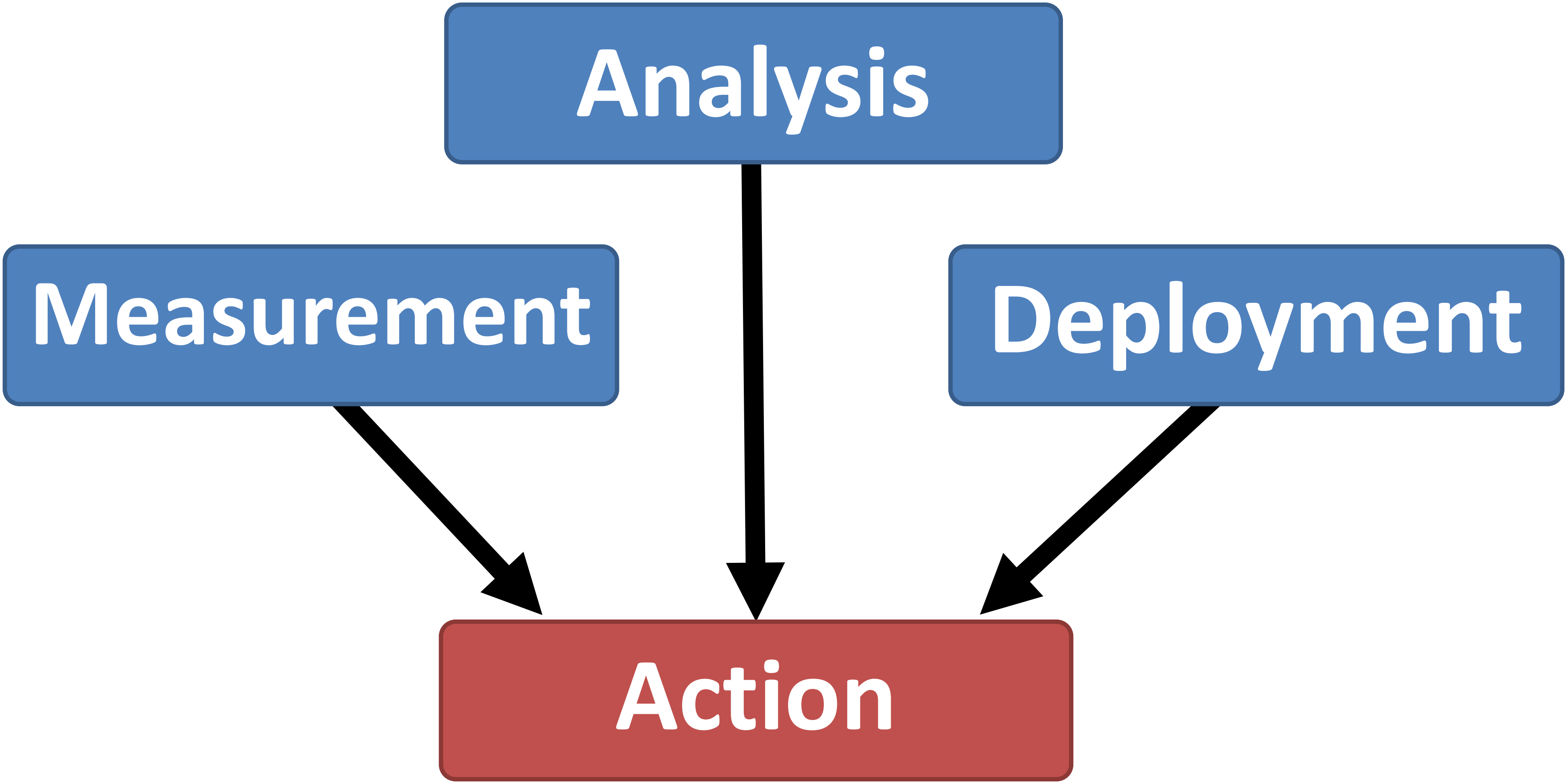
A general approach to performance optimization

- Developer describes application structure via simple interface
  - Identifies tasks
  - Tasks: provides numbers that correlate with compute time (e.g. #non-zeros)
  - Task pairs: provides numbers that correlate with communication time (e.g. #columns)

Easy to use
Like a type system

- Separate statistical framework measures, models and schedules in data-sensitive manner
- Single system takes into account both application and system properties

Beyond use-cases: Building strong pillars for real-world intelligent applications

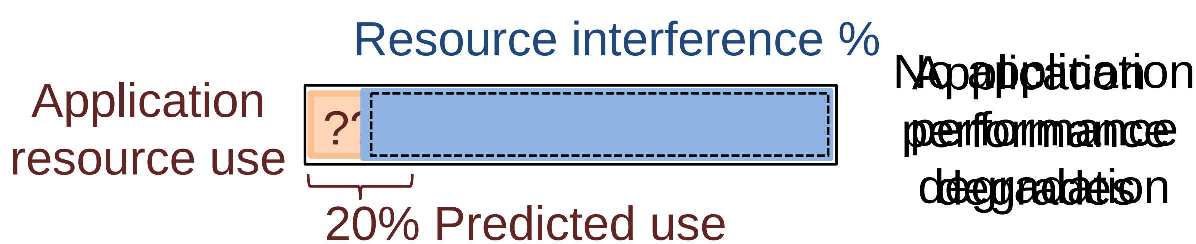


Developing more actionable measurements of application behavior

- Can easily measure execution time, or performance counters (e.g. cache misses)
- Information can not answer basic questions:

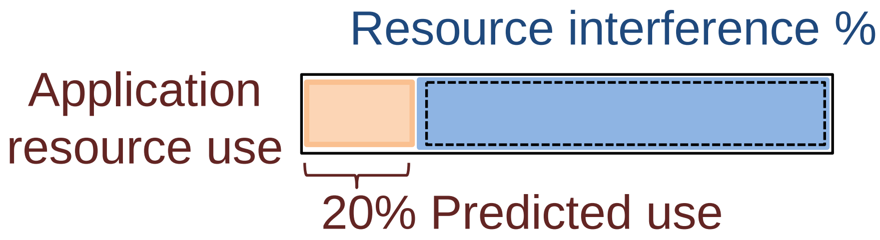
  “If threads A and B run on same core how much will they slow down?”

- Developing measurements that capture application utilization of system



Developing more actionable measurements of application behavior

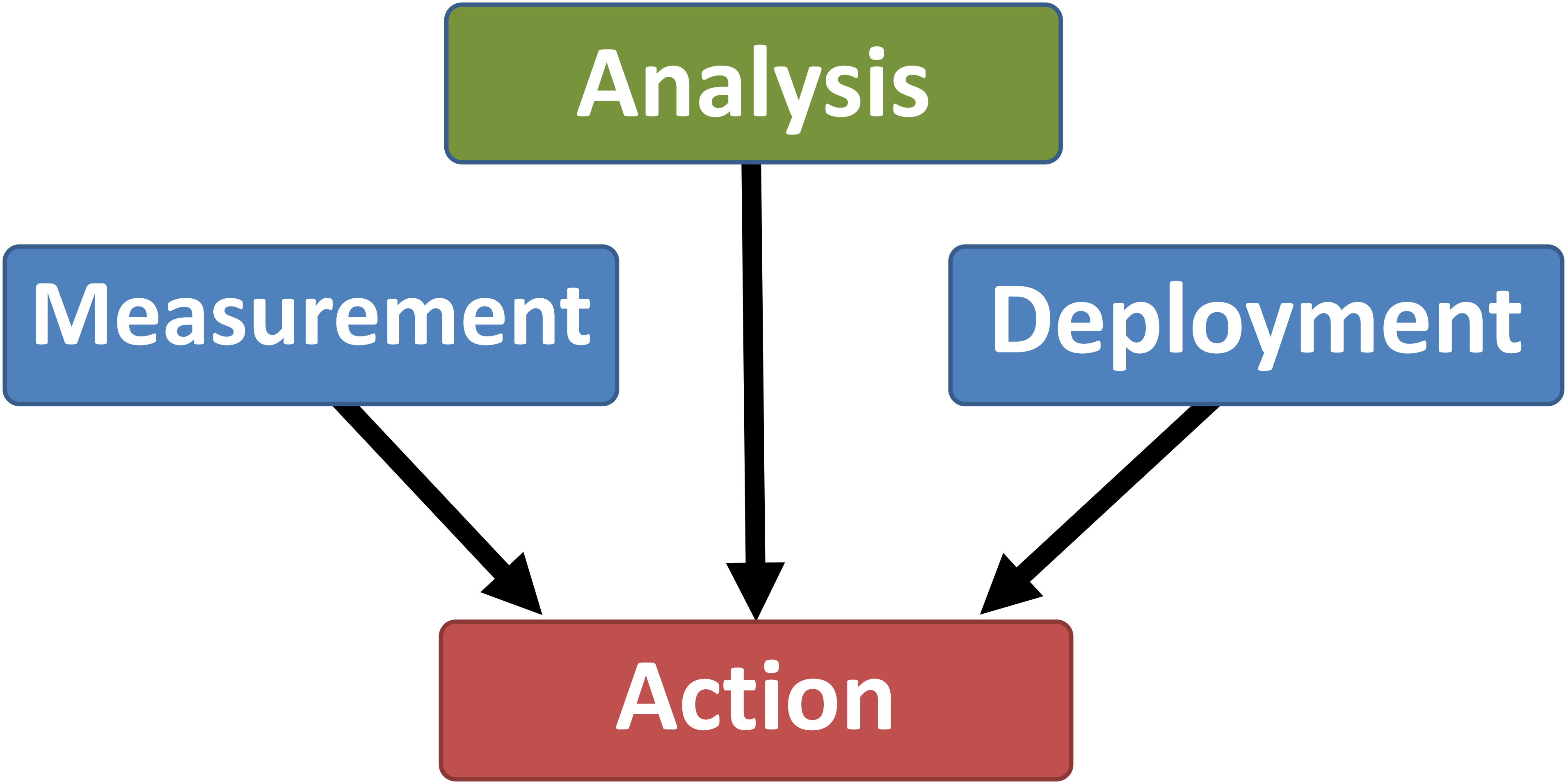
- Developing measurements that capture application utilization of system



- Currently support: storage and bandwidth of shared caches, network bandwidth

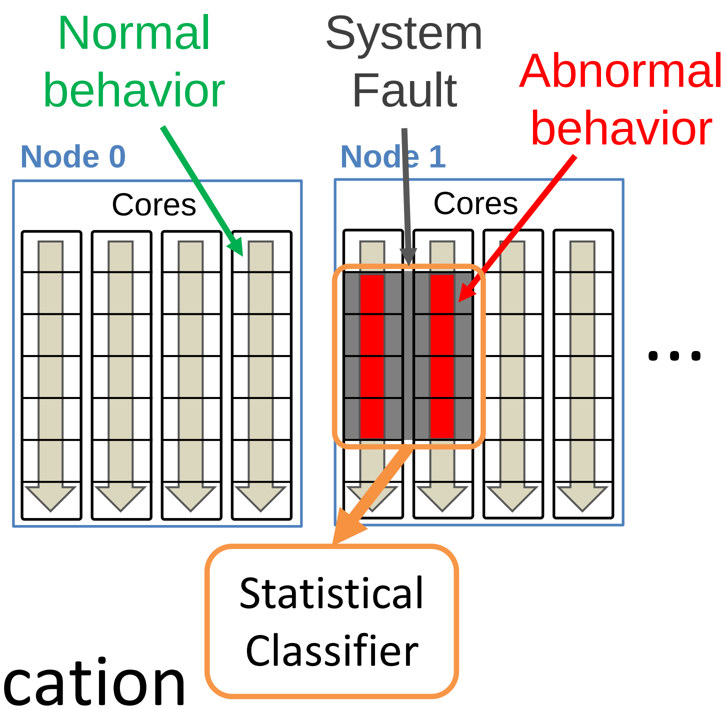
- In development: CPU resources, file system

Building strong pillars for real-world intelligent applications



Detection and localization of system faults requires precise models of application behavior

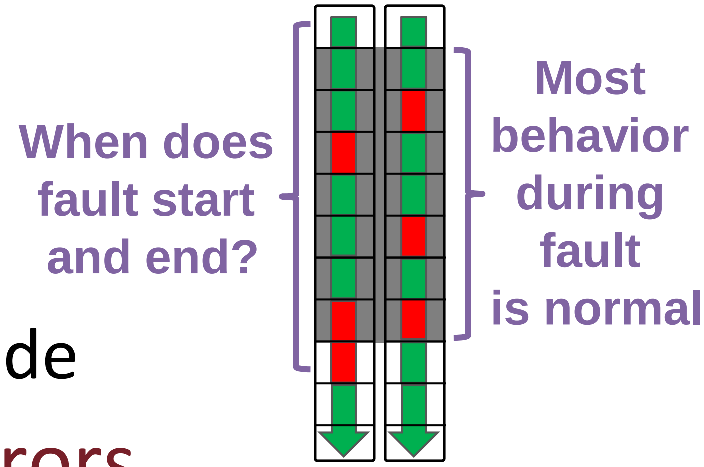
- Fault affects a fraction of application threads
- To detect fault type (CPU, Memory, Network)
  - Inject known fault types into application
  - Train statistical classifier on application behavior during each
  - Predict type based on application behavior in production



Application response to faults depends on unknown factors

- In reality faults have inconsistent effect on applications
  - CPU slowdowns only affect CPU-intensive code regions
  - Errant daemons primarily affect concurrently running code
- Difficult to detect, localize errors
- Characterization becomes very hard

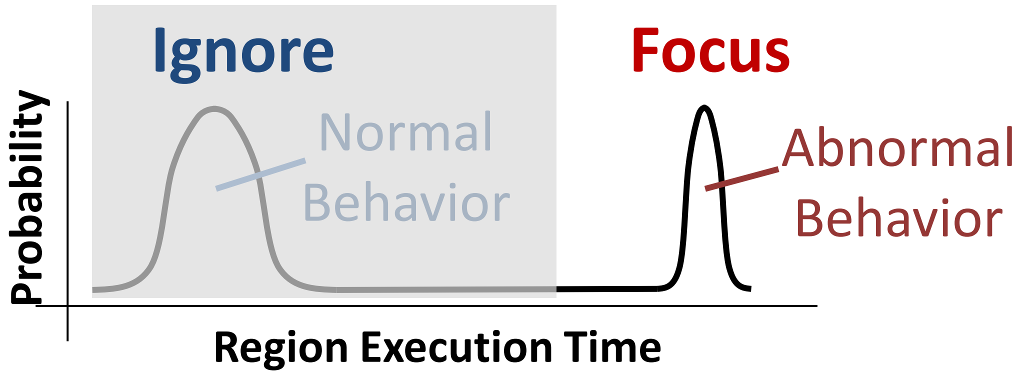
Even if fault duration is known during training most events are not representative of fault



Can improve accuracy by extracting hidden influence from observations

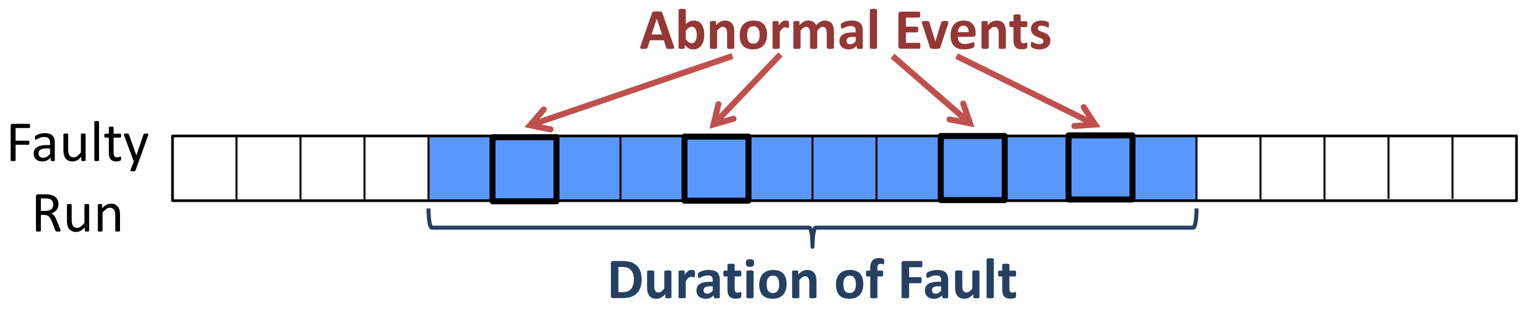
- During fault only few events are abnormal
- Only these truly represent the fault
  - Filter events in faulty runs to only train statistical model on abnormal events
  - Ignore others

Model Trained for Given Code Region

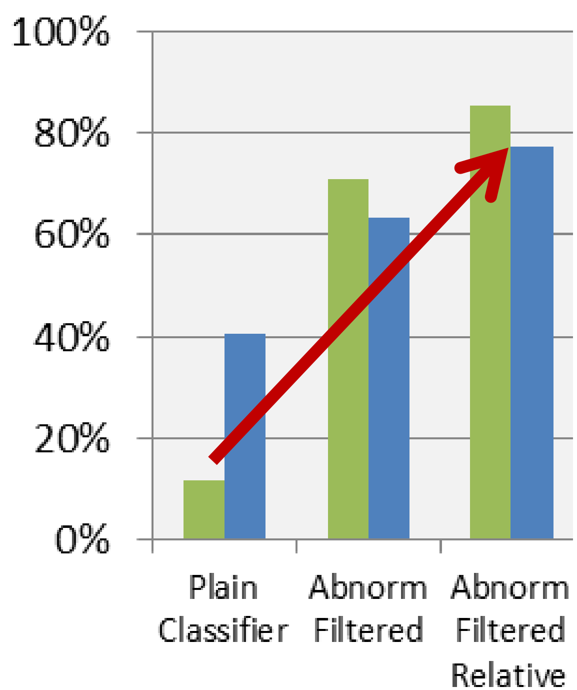


Can improve accuracy by extracting hidden influence from observations

- During fault only few events are abnormal
- Only these truly represent the fault
  - Filter events in faulty runs to only train statistical model on abnormal events
  - Ignore others

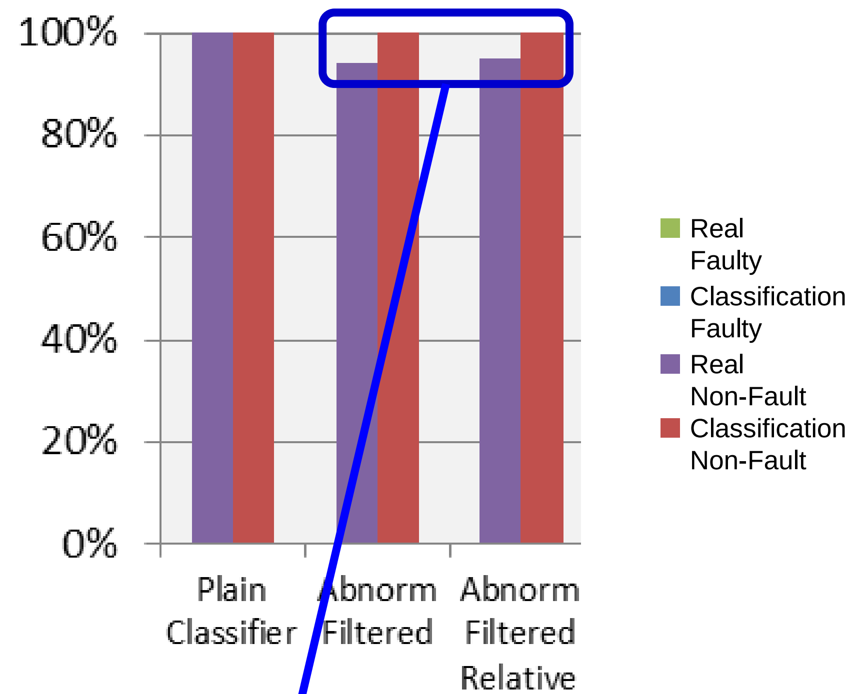


Improved technique detects fault's location, time and type



Faulty Runs

Significant increase in rate of fault characterization

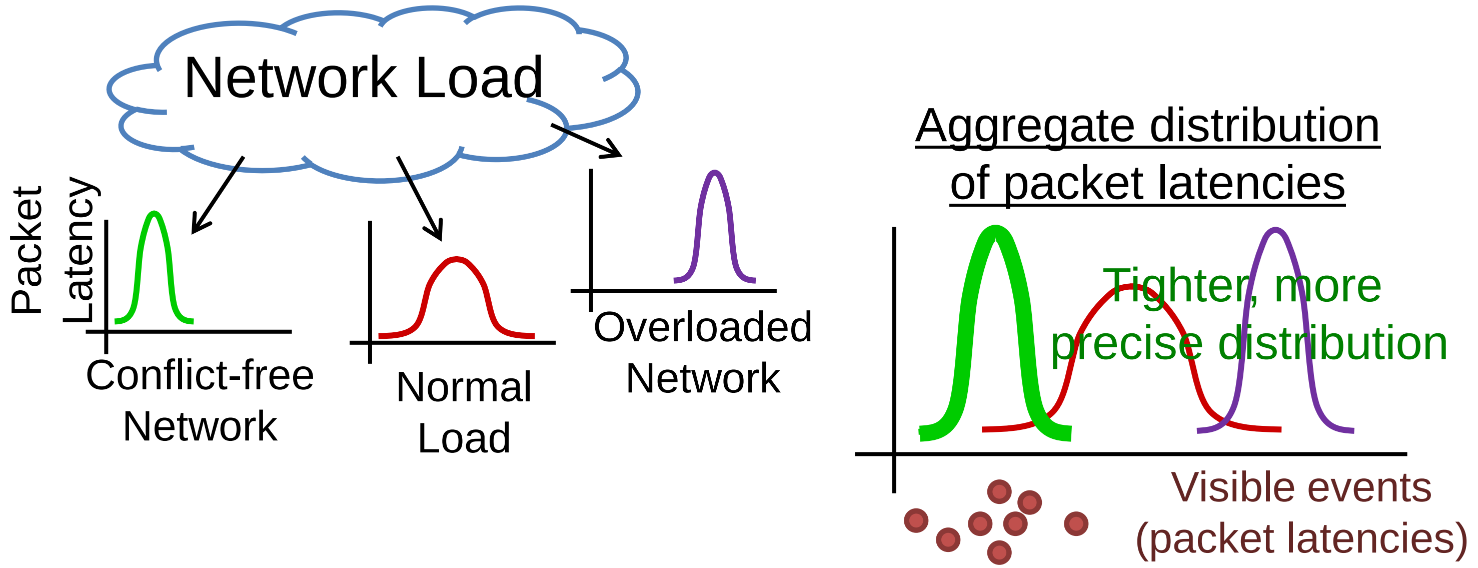


Non-Faulty Runs

Small increase in false positive rate

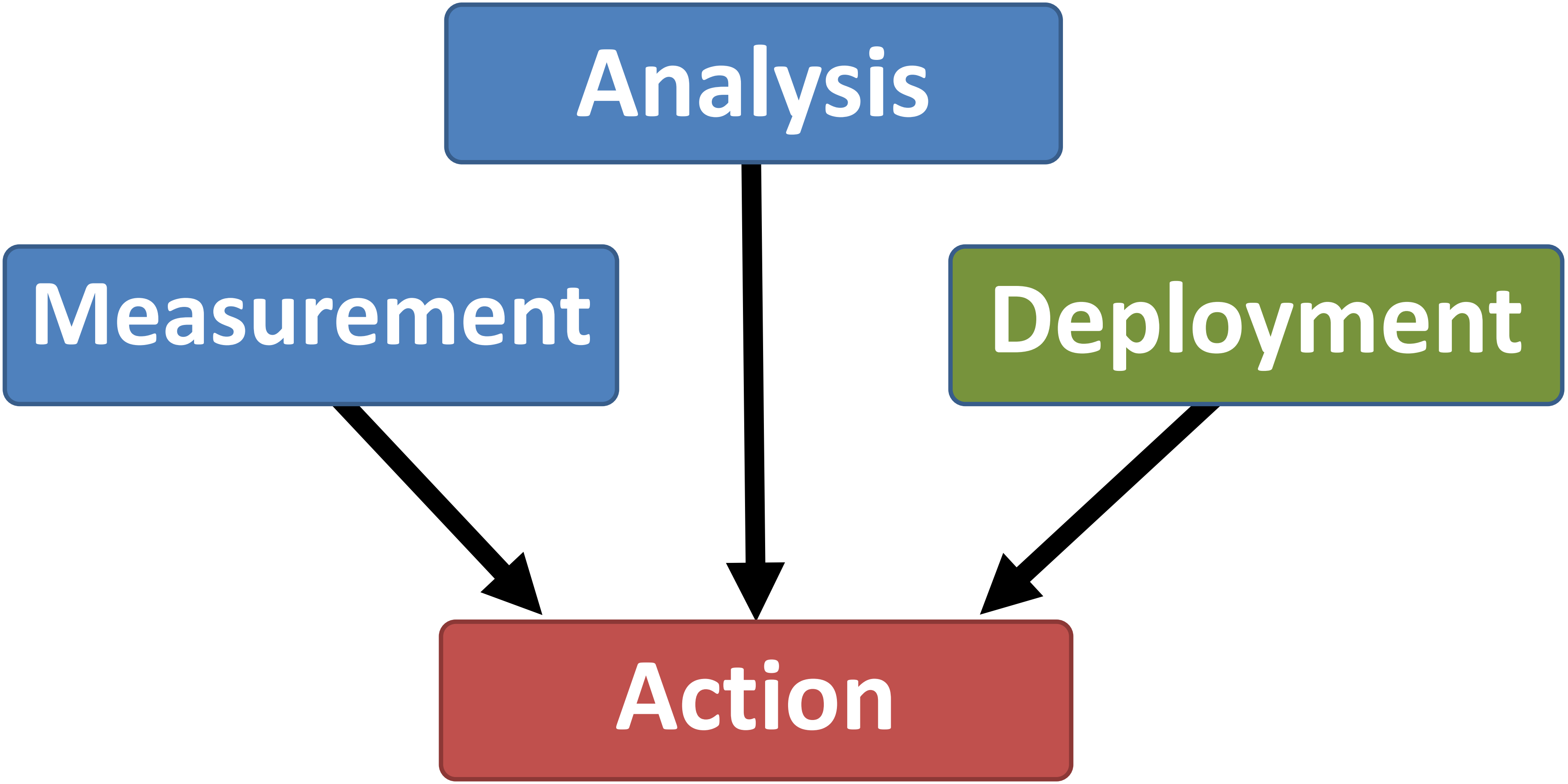
Hidden variable inference: general approach to modeling behavior of complex systems

- System behavior depends on hidden state



- Infer system state from observed events
- Can predict future events more accurately

Building strong pillars for real-world intelligent applications

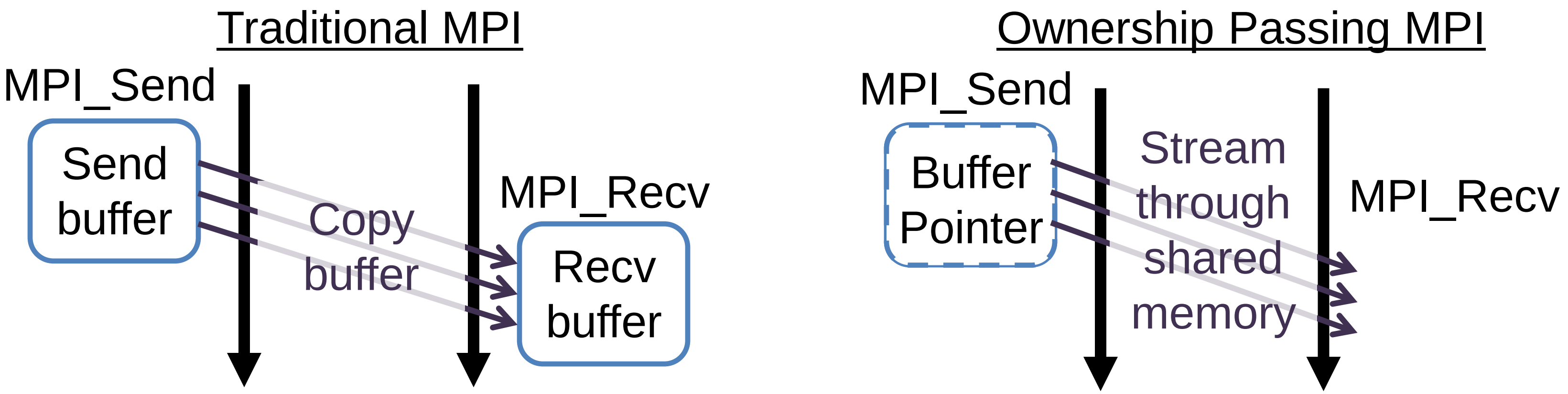


Compiler analyses enable more complex optimizations

- Modeling makes it possible to exploit and manage existing application flexibility
- Compiler transformations can create new flexibility
- Developing compiler analyses to enable library-specific transformations
- Current focus: MPI applications

Exploiting full capabilities of libraries requires library-aware transformations

- Developed MPI for shared memory hardware
  - Implements MPI ranks as threads
  - Communicates via direct copies or passing pointers



- Developing analysis in ROSE compiler to transform legacy MPI code to extended API

Developing compositional symbolic analysis framework to enable aggressive optimizations

- Our analyses must run on real applications
  - Very complex control flow and data management
  - Requires multiple analyses to disambiguate common expressions (e.g. *p, a[i*c])
  - Writing, combining all required analyses beyond capabilities of individual research groups
- We are developing a new compositional symbolic analysis framework
- Enables analyses to use each others' results without knowledge of APIs or abstractions

Example: analyzing even simple programs requires multiple analyses

- Client analysis needs the value printed

Original Program → **Value Analysis** → **Points-To Analysis** → **Value Analysis**

Original Program

```
int x=5;
int y=12;
int* p;

if(x<y) p = &x;
else p = &y;

print *p+5;
```

Value Analysis

```
int x=5;
int y=12;
int* p;
if(TRUE) p = &x;
else p = &y;
print *p+5;
```

Points-To Analysis

```
int x=5;
int y=12;
int* p;
p = &x;
print x+5;
```

Value Analysis

```
int x=5;
int y=12;
int* p;
print 10;
```

- Composition of independent analyses enables complex transformations of real applications

Analysis

- Dynamic statistical modeling
- Sensitive to properties of input data, system, hidden state

Measurement

- Application behavior, not system metrics
- Quantify algorithm accuracy

Deployment

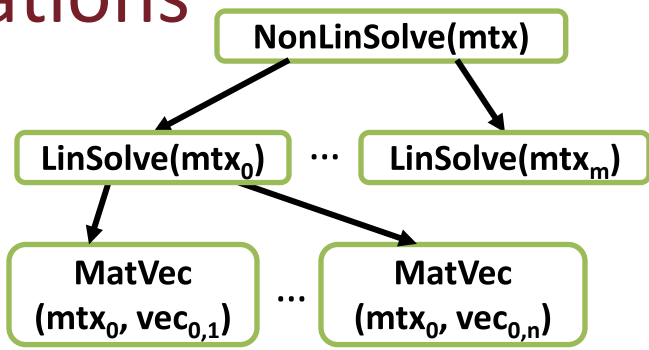
- Include adaptivity and measurement into applications
- Transform code to use new optimizations

Action

- Configure application and system based on analysis results
- Guide developers, system administrators

Prototyping modeling and adaptivity research in custom-designed work manager

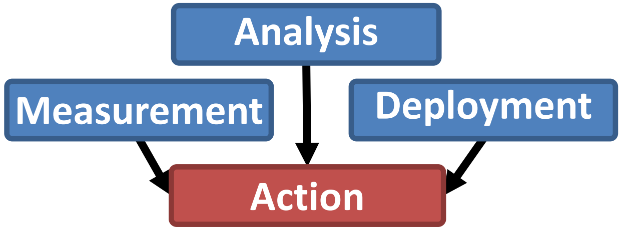
- It is very complex to incorporate model guidance into existing runtimes
- Have developed a custom work manager to prototype optimizations
  - Explicit tasks and dependencies
  - Easy to incorporate new models and dynamic optimizations



- Results directly applicable to real-world runtime systems (e.g. Charm++, Hadoop)

Application Behavior Analysis enables productive use of complex applications and systems

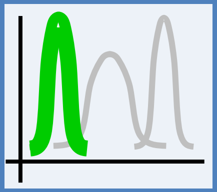
- Severe resource constraints force applications and systems to become more flexible and complex



- Behavior analysis and modeling required to productively use applications and systems
  - Have demonstrated utility of approach in representative use-cases
  - Ongoing research on increasing capability and generality of approach

Analysis

Discovering hidden system state from observations



Measurement

Measuring application resource use

Deployment

Composable symbolic compiler analyses

```
int x=5;
int y=12;
int* p;

if(x<y) p = &x;
else p = &y;

print *p+5;
```

Action

Simple testbed workflow manager for application/system co-optimization