# *The Ghost in the Machine*

# Observing the Effects of Kernel Operation on Parallel Application Performance

**Aroon Nataraj, Alan Morris, Allen Malony, Matthew Sottile, Pete Beckman[‡]**

{anataraj, amorris, malony, matt}@cs.uoregon.edu

Department of Computer and Information Science

University of Oregon

beckman@mcs.anl.gov[‡]

Mathematics and Computer Science Division

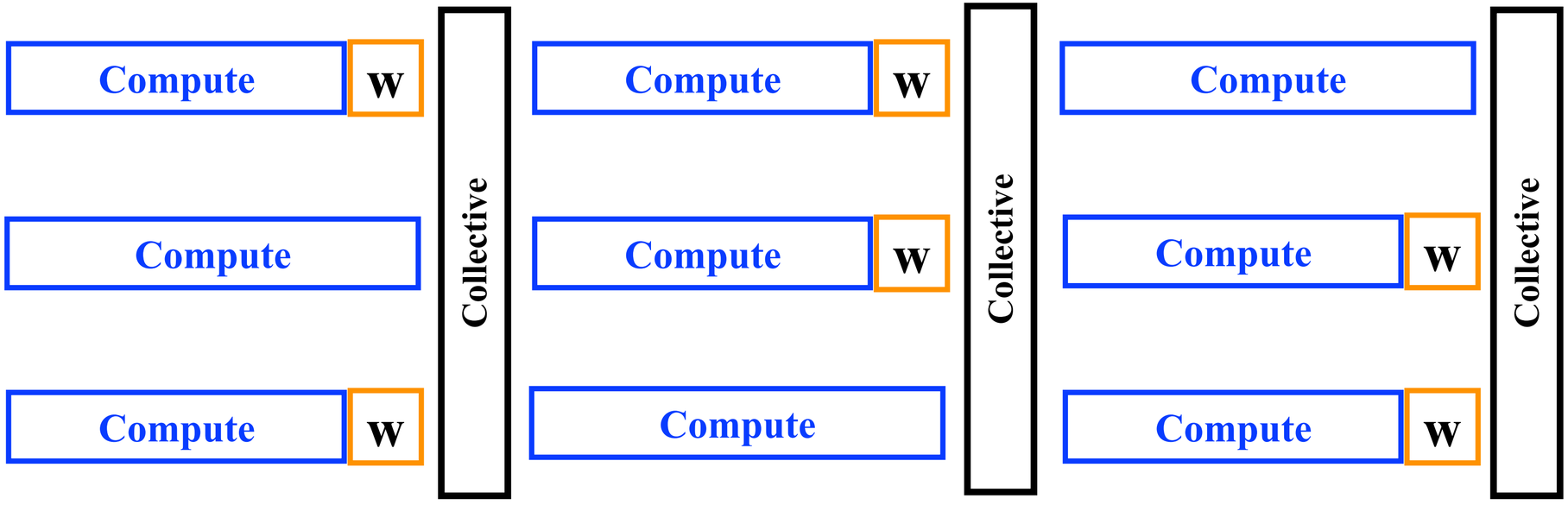
Argonne National Laboratory

# Talk Outline

- The Problem
  - What is Operating System / Runtime (OS/R) interference
  - Is it a problem?
  - Measurement question
- The Solution
  - KTAU and TAU performance systems
  - Fine-grained OS/App performance correlation
  - Noise-effect estimation technique
- Evaluation
  - Demonstrating measurement and analysis of real OS noise
  - Investigating accuracy of noise-estimation analysis at scale
- Conclusion

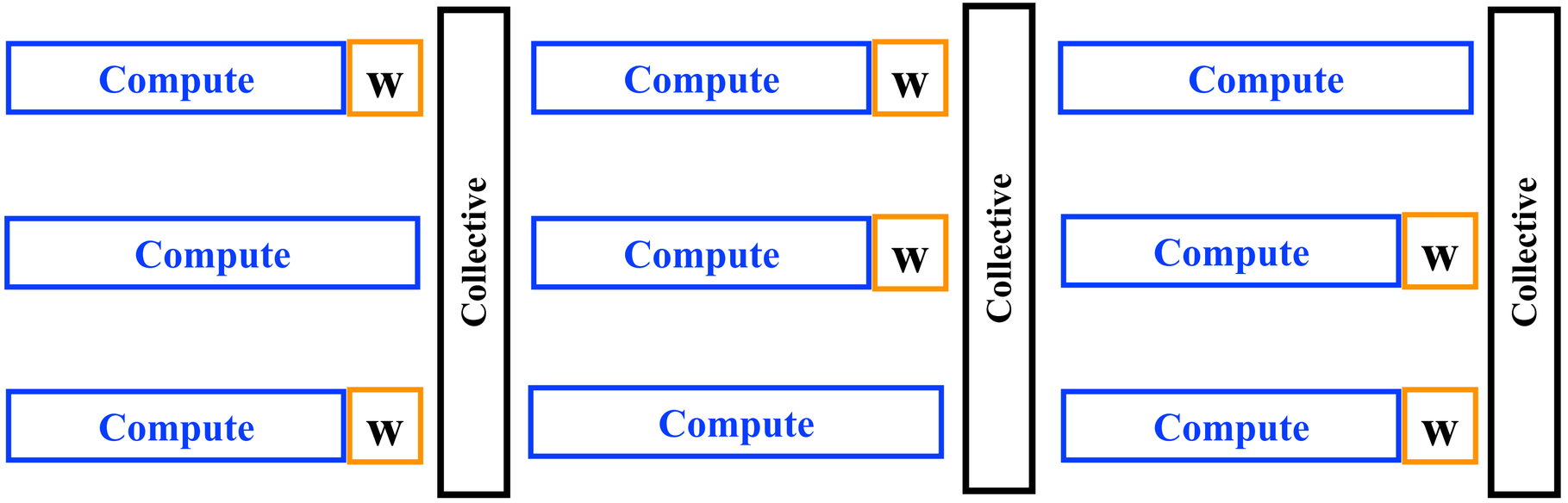
# *Example of Noise and its Propagation*

Phases of Alternating Computation & Collectives



# *Example of Noise and its Propagation*

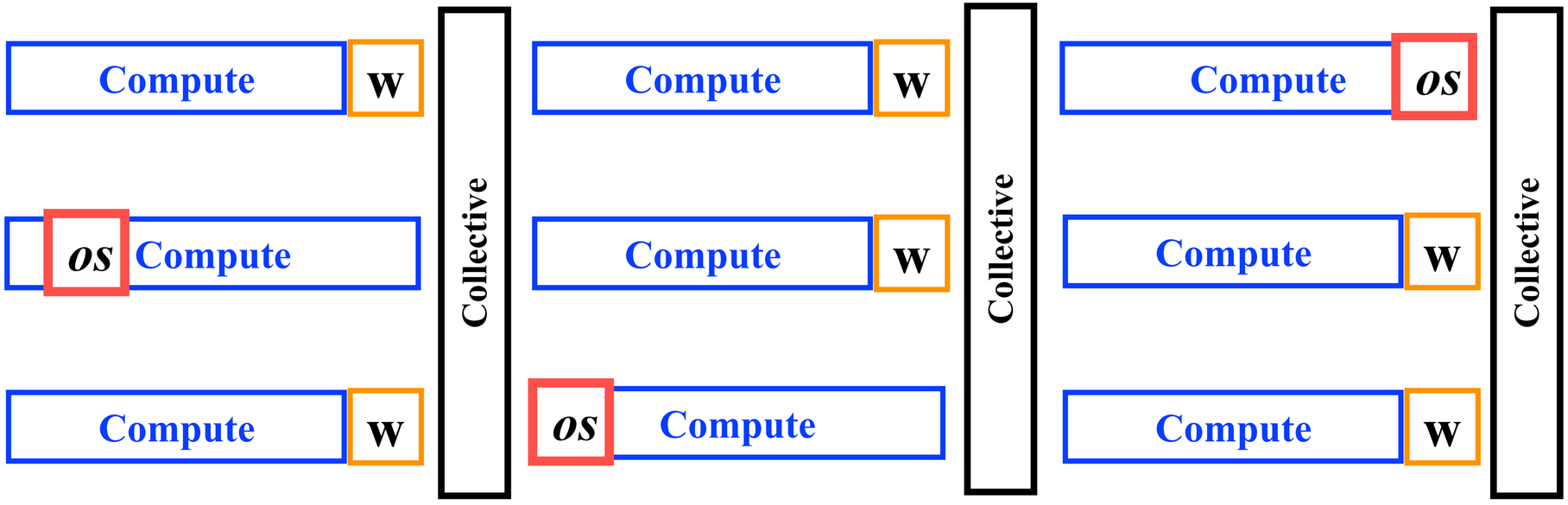
Phases of Alternating Computation & Collectives




**What is the cause of imbalance?**

# *Example of Noise and its Propagation*

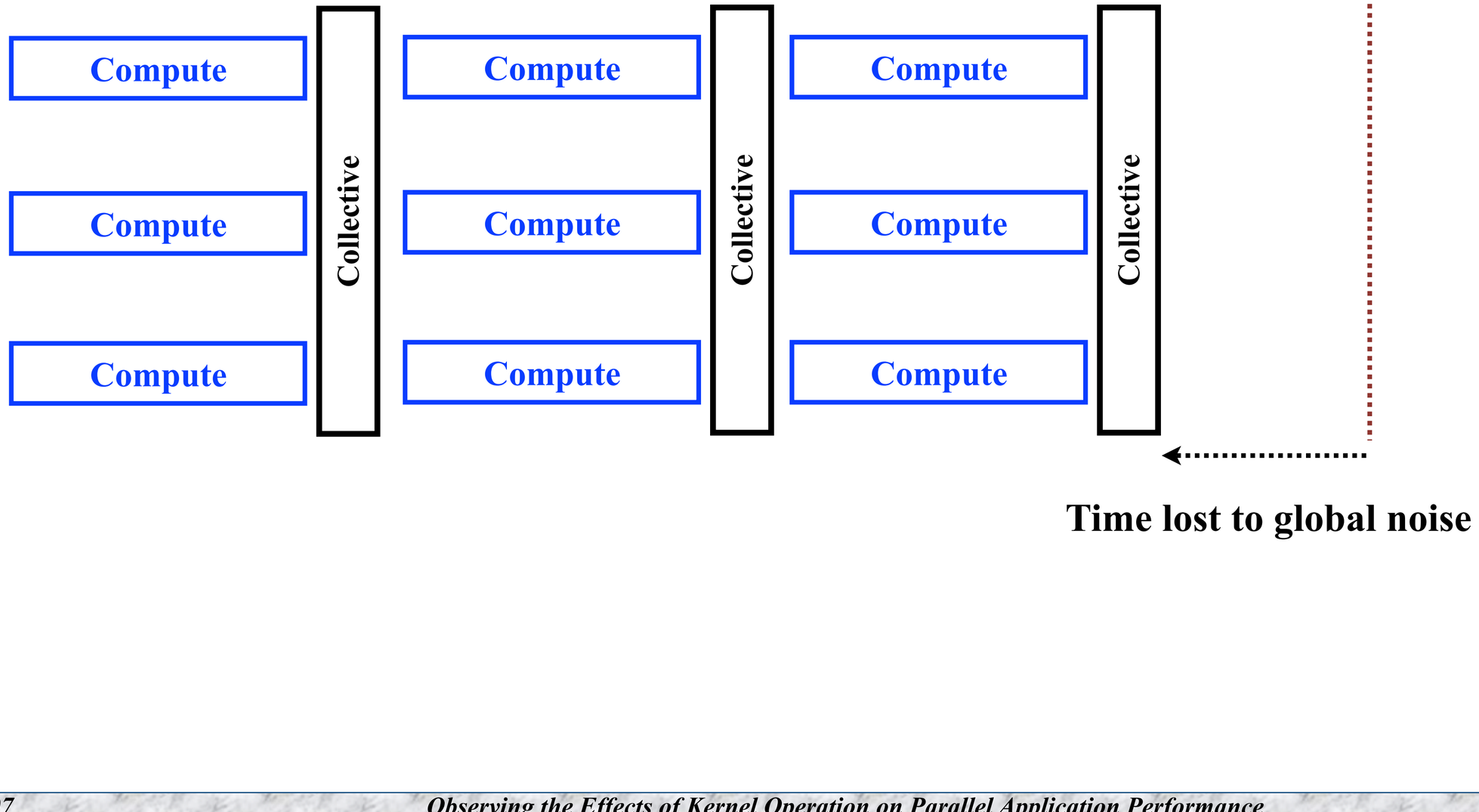
Phases of Alternating Computation & Collectives




**What is the cause of imbalance?**

# Example of Noise and its Propagation

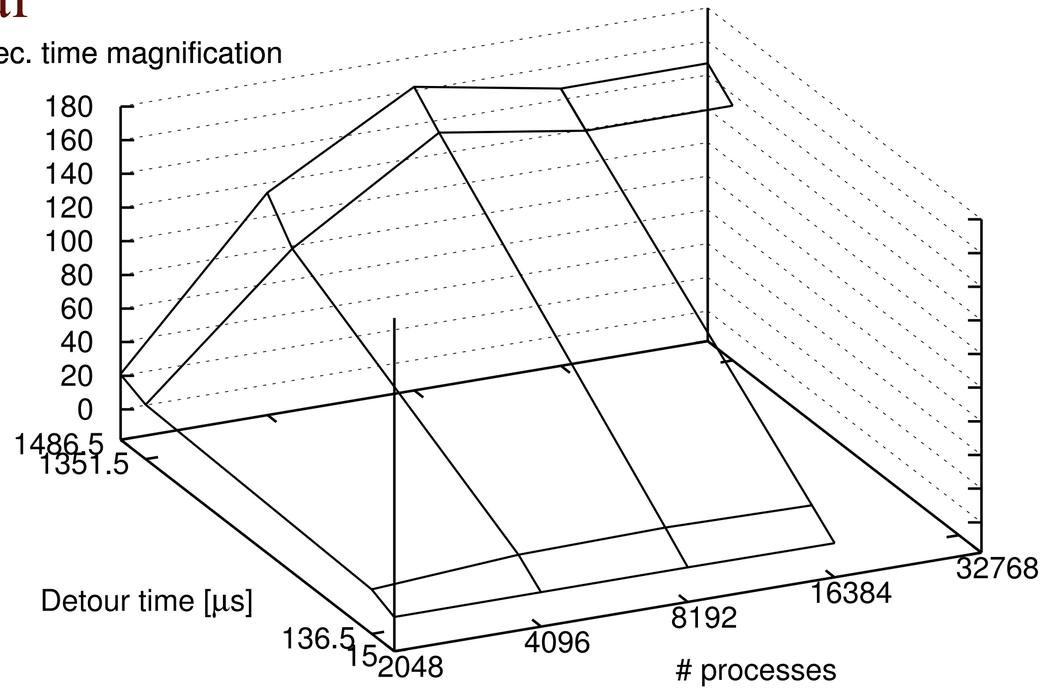
Phases of Alternating Computation & Collectives



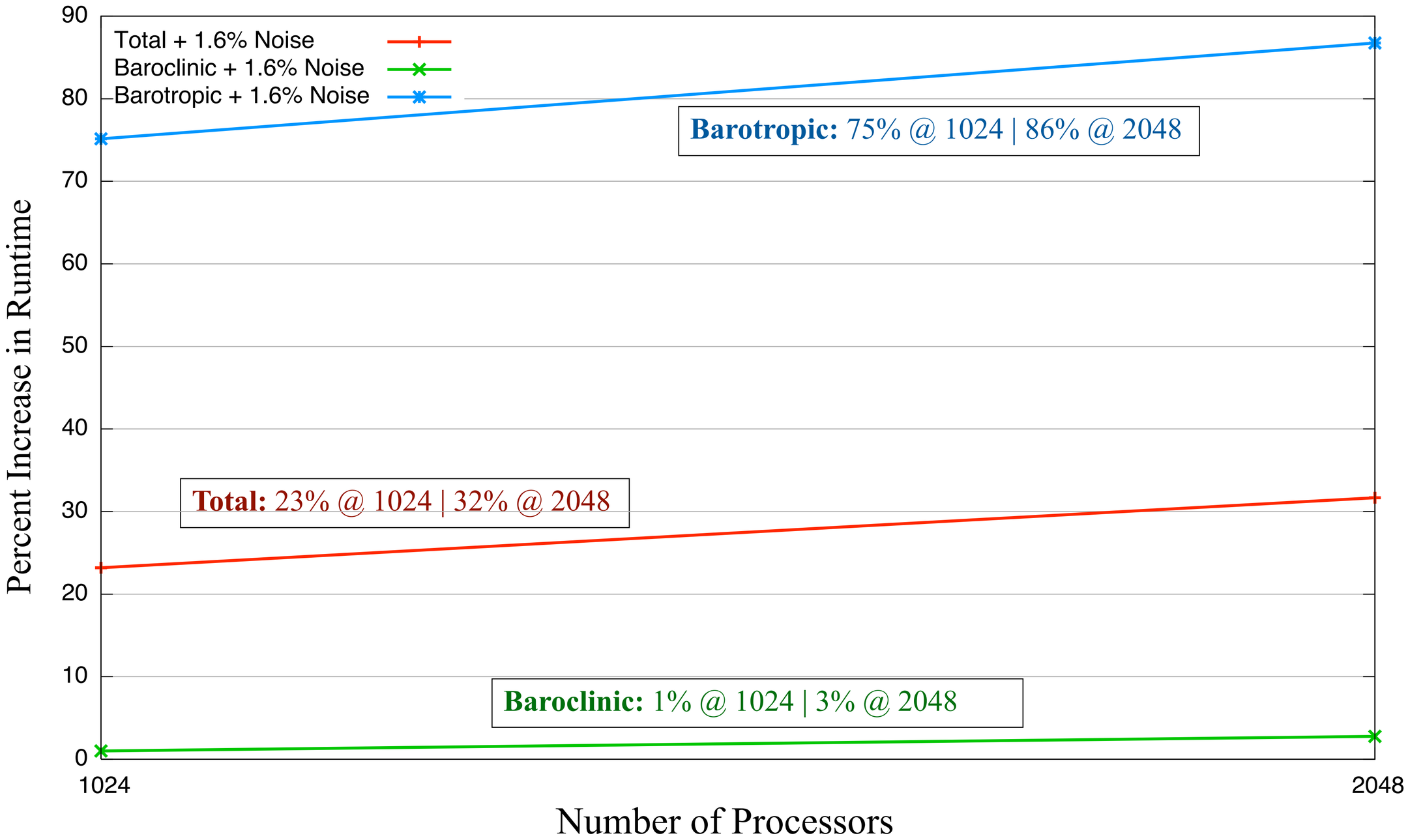
# *Is OS Noise a problem? How?*

- Previous work has shown significant OS/R interference problems
  - Large variability in point-point communication latency [Mraz SC'94]
  - Measurement mismatched performance model [Petrini et al. SC'03]
  - Poor scaling performance of collectives [Jones et al. SC'03]
- Nature of noise matters
  - Theoretical modeling [Agarwal et al. HiPC'05]
  - Noise Emulation [Beckman et al. CCJ'07]
    - Heavy-tailed and Bernoulli noise most detrimental
    - Effect of noise on collectives
    - Maximum noise duration determines effects
  - Large, rare noise-events are problematic

Beckman et al. CCJ'07



# *Effect of Injected Noise on a Real Application (POP)*



# *Noise Effects are Complex*

# *Noise Effects are Complex*

- Local noise dependent on the OS/R and its configuration, but global effects ...
- Depend on the underlying platform
  - Interconnect latency, timer resolution, TLB ...
- Depend on the OS/R configuration and noise sources
  - Scheduling policy, interrupt frequency, daemons ...
- Depend on parallel application behavior
  - Synchronous communications, computational grain, load balance ...

- Can we measure the *global delay* an application experiences due to specific noise sources?
- *Real application + real, existing noise => noise effect?*

# *Our Approach and Contribution*

- General noise-effect estimation by direct measurement of application and OS

- Contribution
  - Isolate OS noise in application performance data
  - Quantify global effects of noise on application
  - Attribute the effects to specific noise sources
  - Analyze application sensitivity

# *How do we measure?*

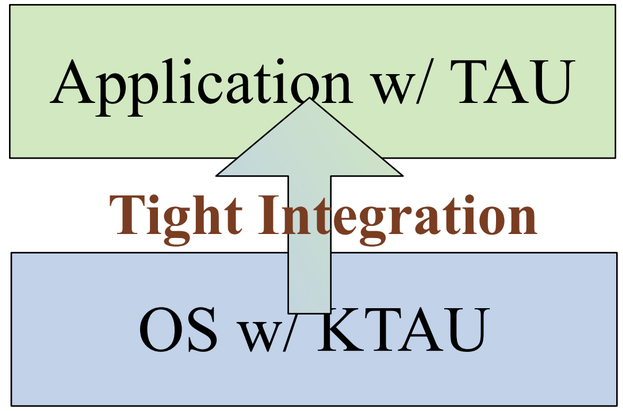
- Application
  - TAU Performance System
  - Profiling and/or tracing of application events
  - Execution time, h/w performance counters ...
- Operating System
  - KTAU
  - Profiling and tracing of system-level events
  - System calls, scheduling, interrupts, ...
- Integration
  - KTAU extended to allow fast access to system performance data
  - TAU captures OS data as counters stored with application events

Application w/ TAU

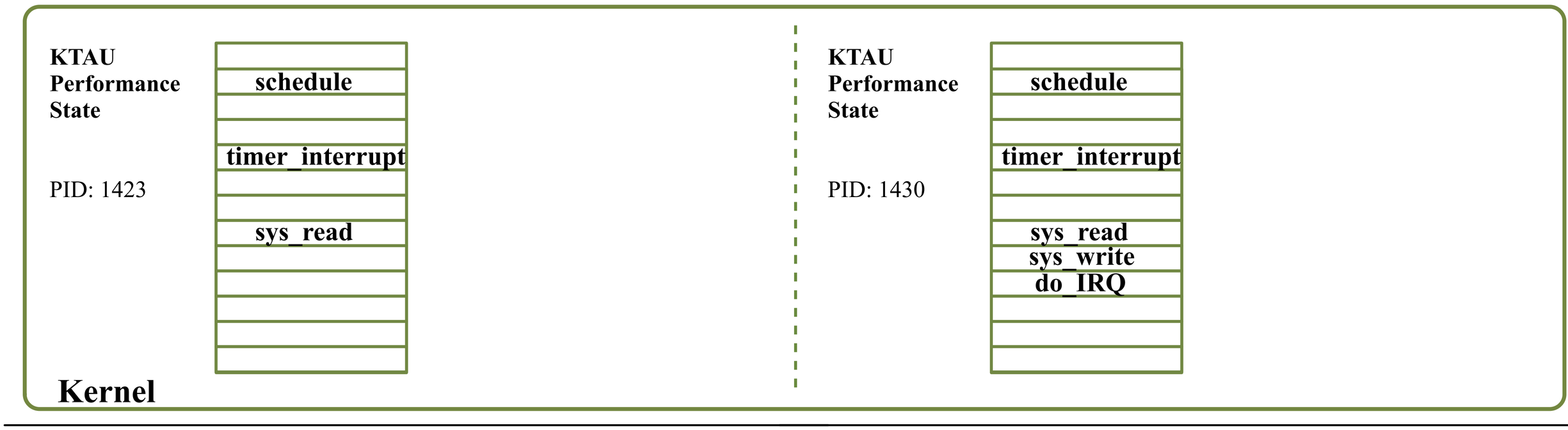
OS w/ KTAU

# *How do we measure?*

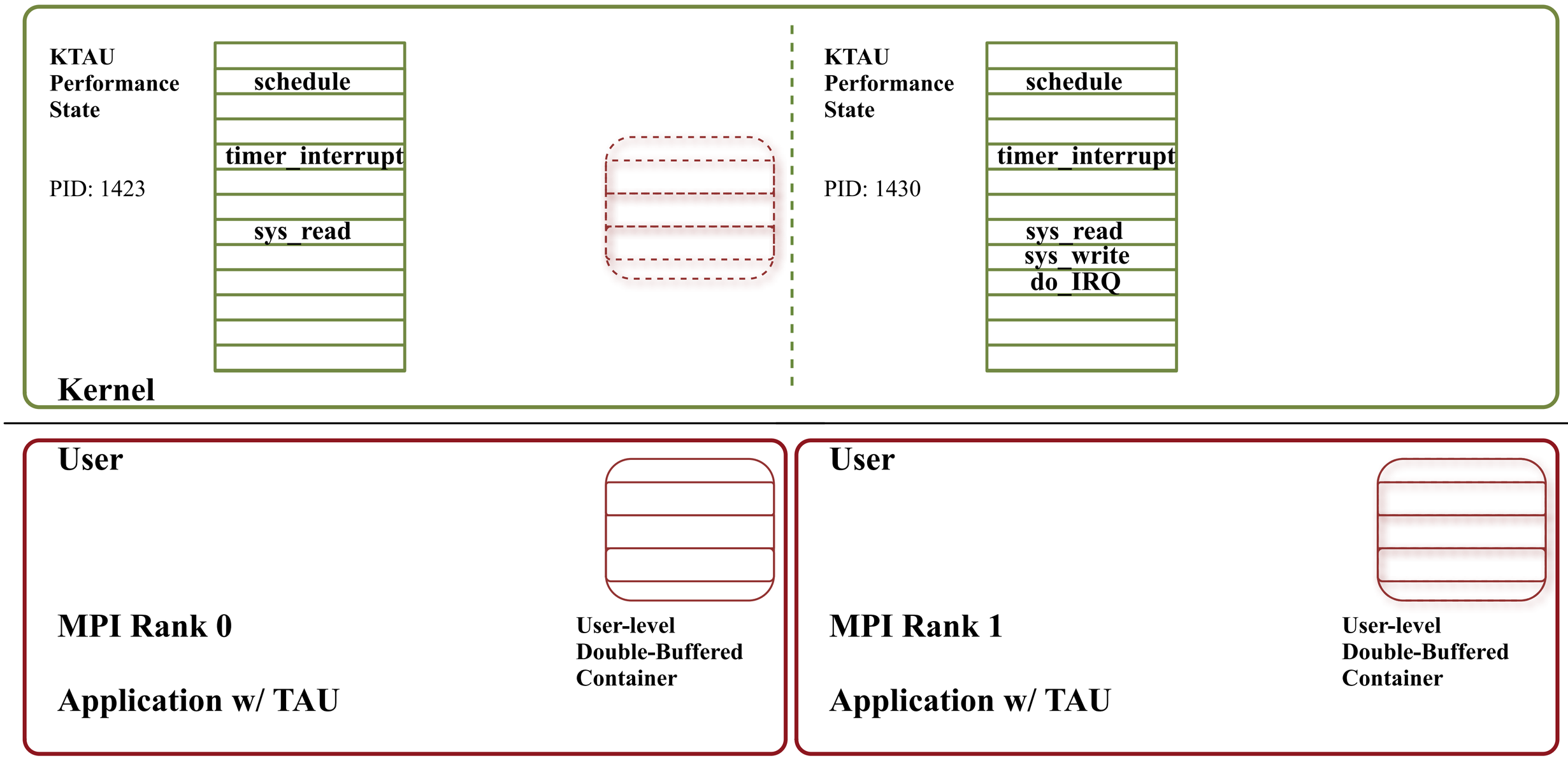
- Application
  - TAU Performance System
  - Profiling and/or tracing of application events
  - Execution time, h/w performance counters ...
- Operating System
  - KTAU
  - Profiling and tracing of system-level events
  - System calls, scheduling, interrupts, ...
- Integration
  - KTAU extended to allow fast access to system performance data
  - TAU captures OS data as counters stored with application events



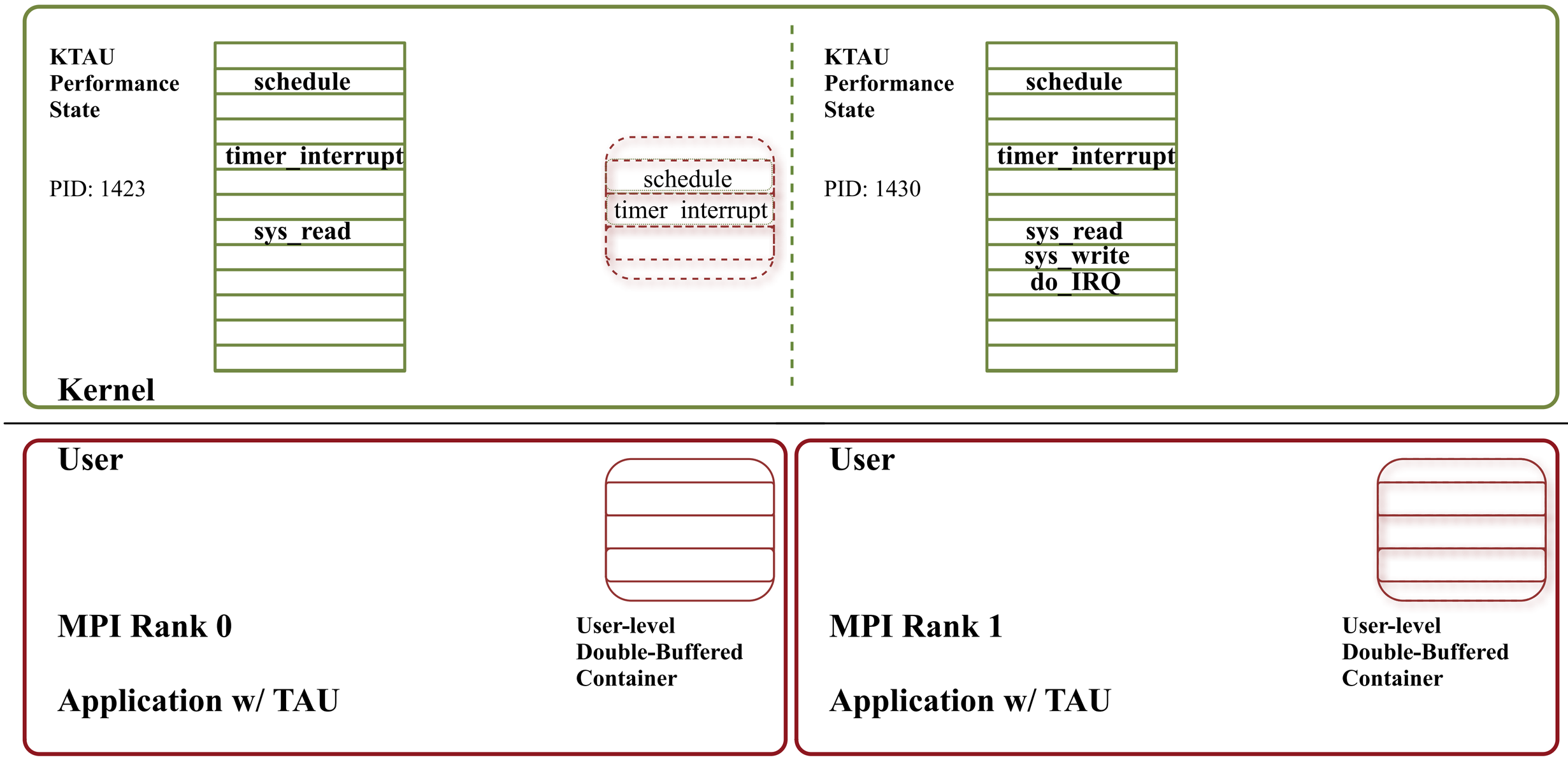
# How the integration works...



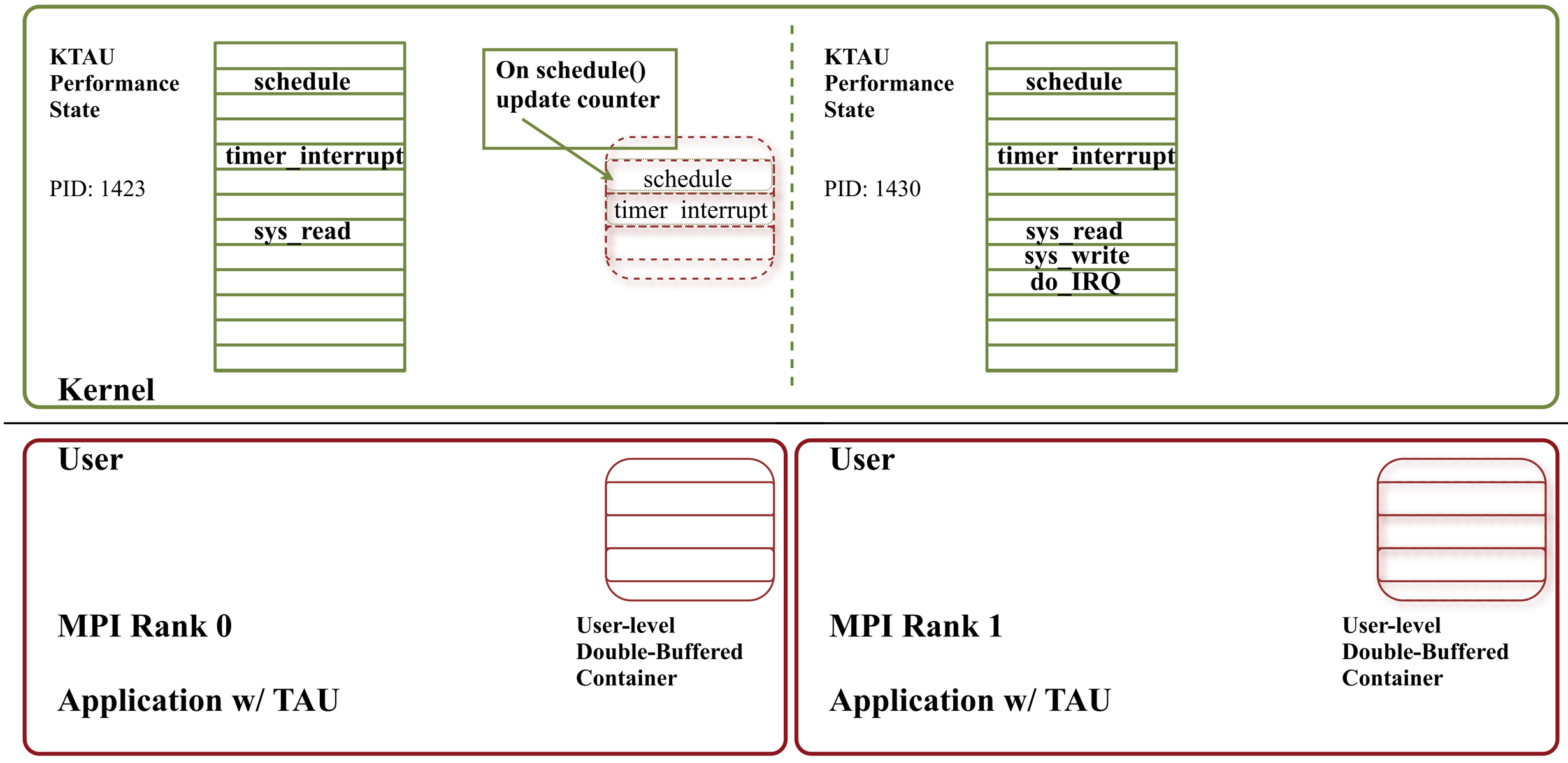
# How the integration works...



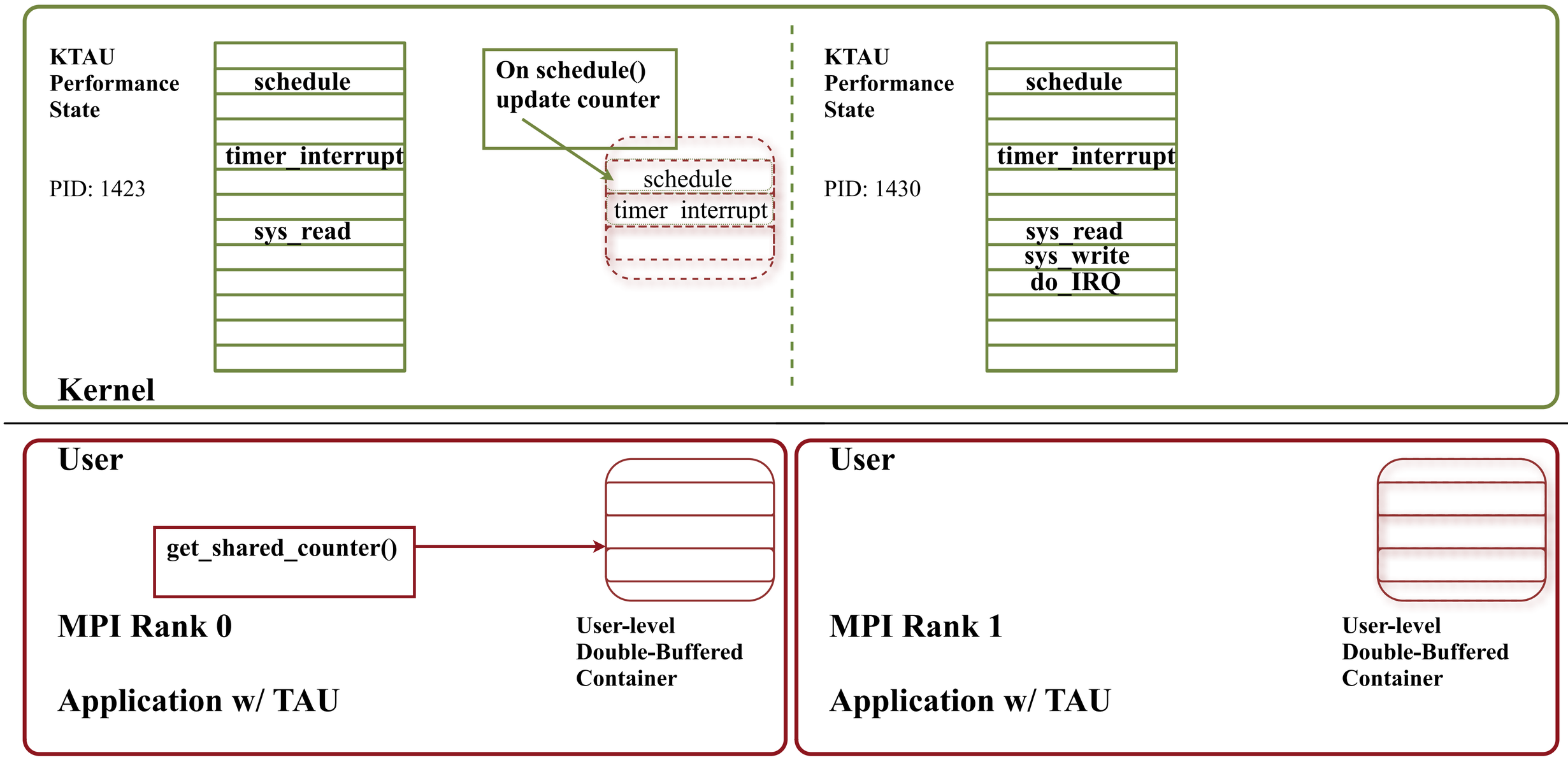
# How the integration works...



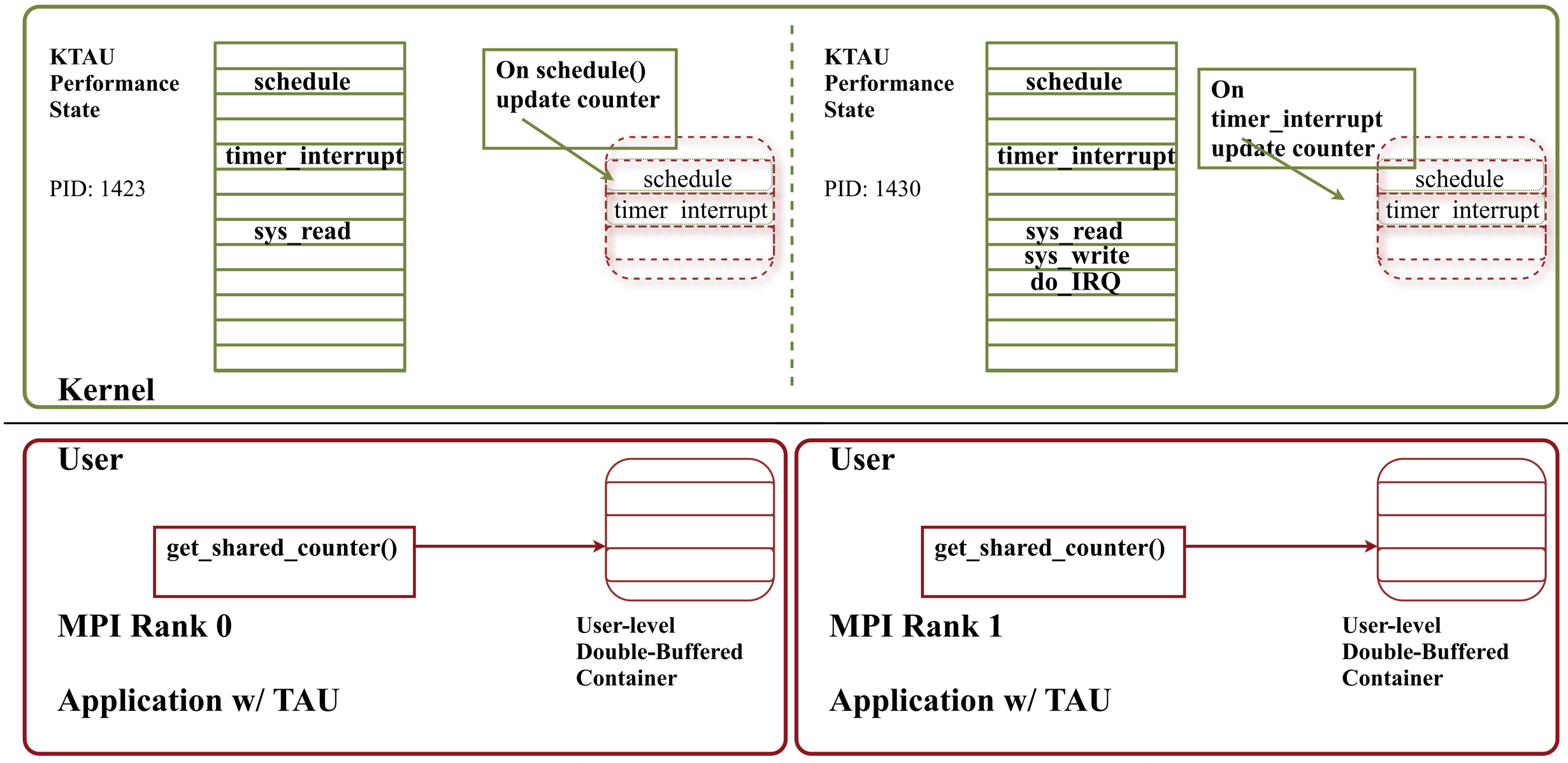
# How the integration works...



# How the integration works...



# How the integration works...




**Per-Process Virtualized OS Counters**

**No Daemon or System Call needed!**

# *How do we analyze?*

- Trace information
  - Application event trace with OS noise counters
  - Presence of OS noise has affected the timing of events
- Timeline Approximation
  - Reason about timing of events that may have occurred in the absence of noise
  - Remove time-duration of noise and adjust event timestamps
  - Must be careful to maintain constraints in event ordering
    - E.g. A recv cannot end before the corresponding send
- Delay due to the global noise-effect (Accumulated Noise)
  - Difference in end times between measured and approximated timeline
- Two main cases
  - Noise Propagation; Noise Absorption
  - See paper for other cases

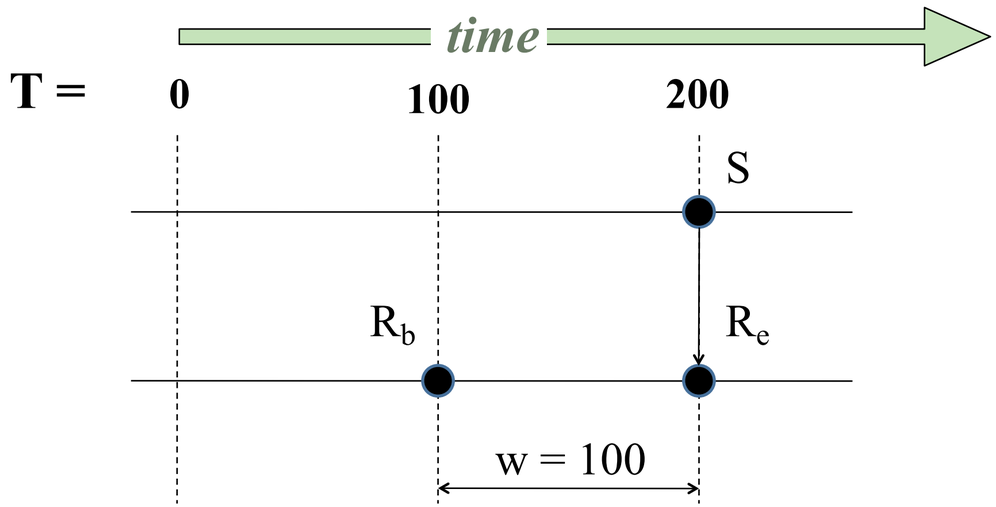
# Global Noise Estimation - Noise Propagation

## Rank 1 Sends; Rank 2 Receives

Measured Timeline

**Rank 1**
***Local* Noise = 100s**

**Rank 2**
***Local* Noise = 0s**



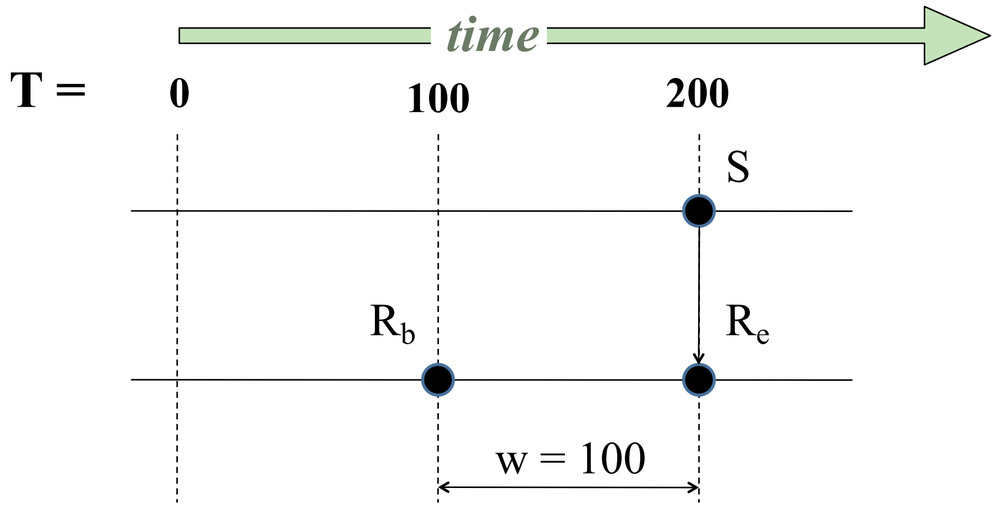
# Global Noise Estimation - Noise Propagation

## Rank 1 Sends; Rank 2 Receives

### Measured Timeline

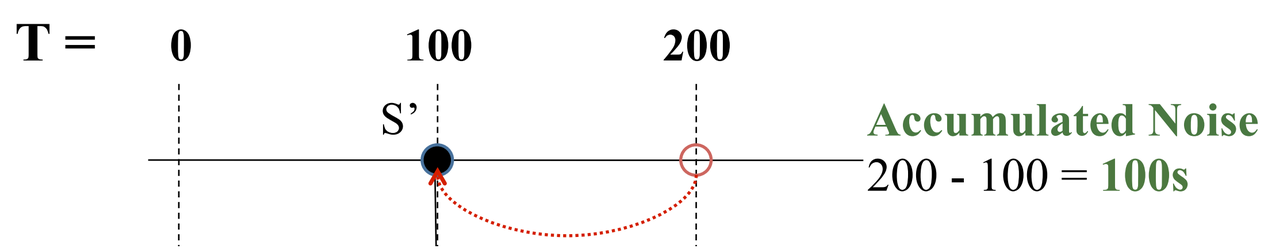
**Rank 1**
***Local* Noise = 100s**

**Rank 2**
***Local* Noise = 0s**




### Approximated Timeline

**Rank 1**
***Local* Noise = 100s**



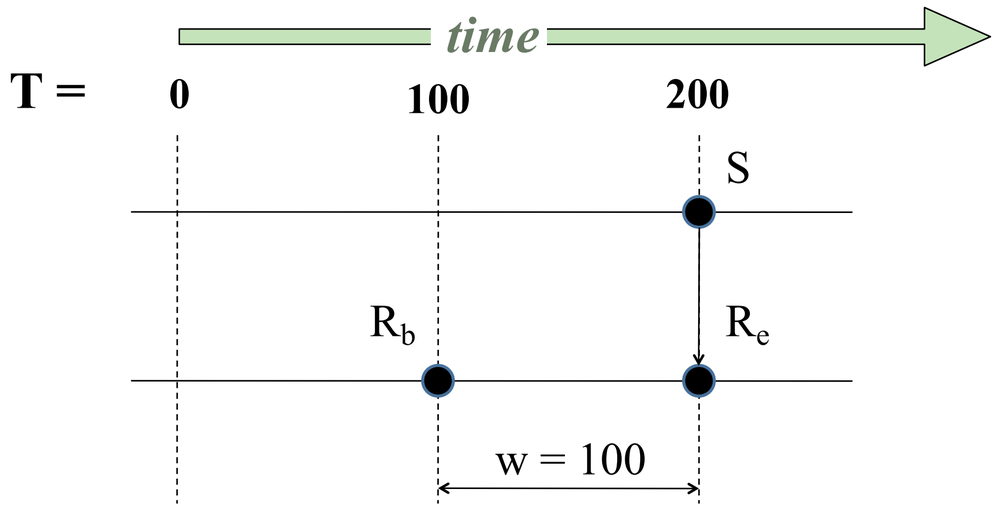
# *Global Noise Estimation - Noise Propagation*

## Rank 1 Sends; Rank 2 Receives

### Measured Timeline

**Rank 1**
***Local* Noise = 100s**

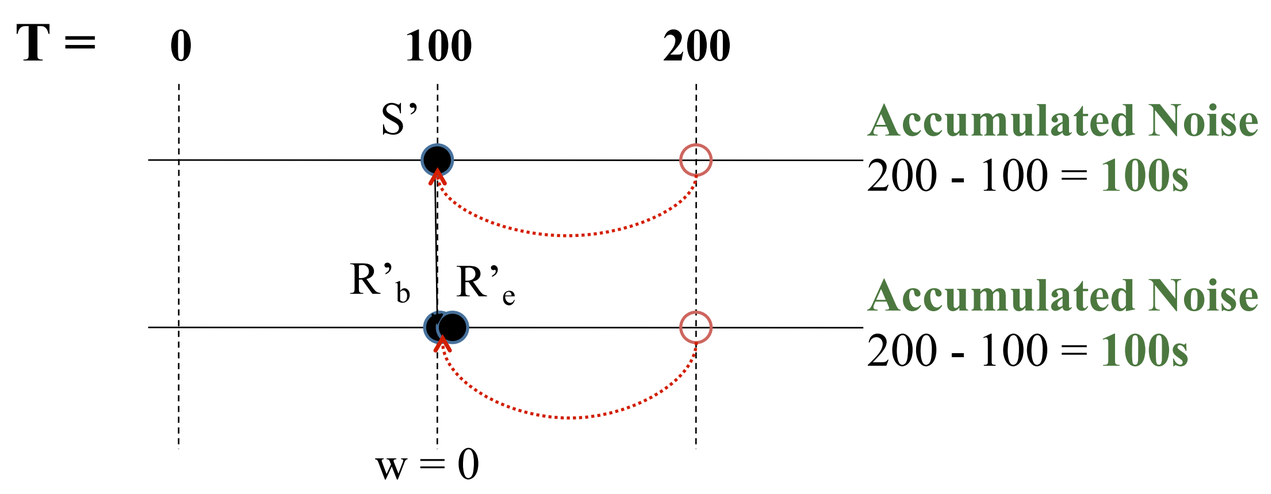
**Rank 2**
***Local* Noise = 0s**




### Approximated Timeline

**Rank 1**
***Local* Noise = 100s**

**Rank 2**
***Local* Noise = 0s**



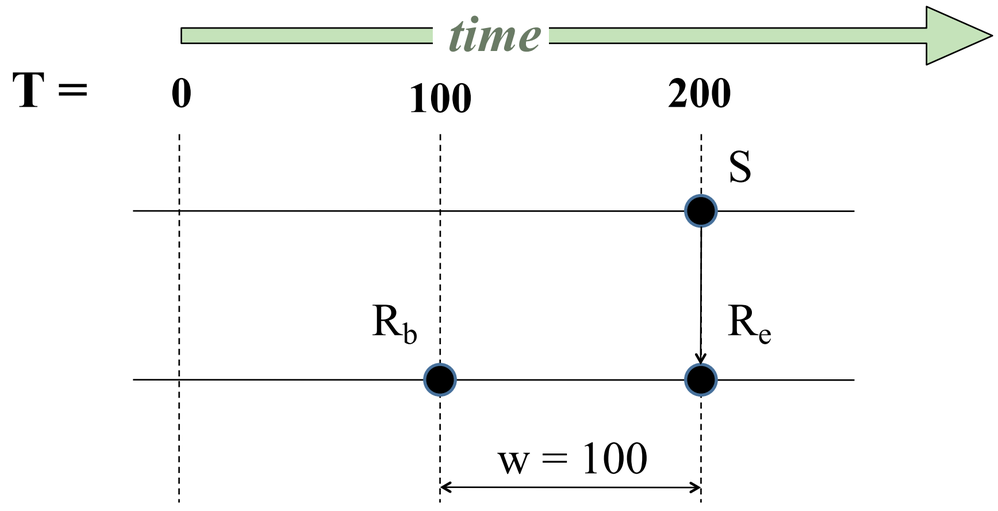
# Global Noise Estimation - Noise Propagation

Rank 1 Sends; Rank 2 Receives

Measured Timeline

**Rank 1**
***Local* Noise = 100s**

**Rank 2**
***Local* Noise = 0s**

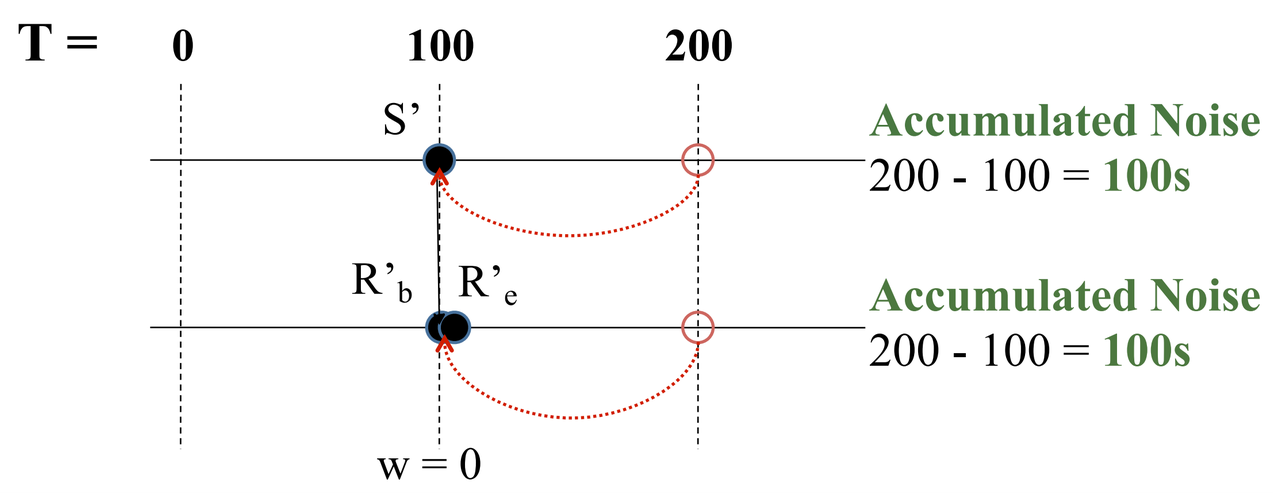



***Accumulated Noise* = Last *Measured* Timestamp - Last *Approximated* Timestamp**

Approximated Timeline

**Rank 1**
***Local* Noise = 100s**

**Rank 2**
***Local* Noise = 0s**



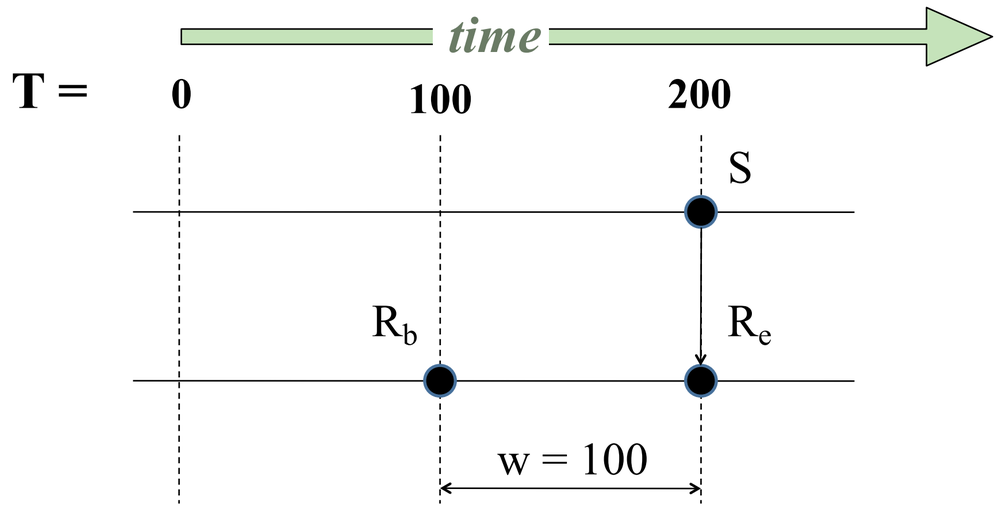
# *Global Noise Estimation - Noise Propagation*

## Rank 1 Sends; Rank 2 Receives

### Measured Timeline

**Rank 1**
***Local* Noise = 100s**

**Rank 2**
***Local* Noise = 0s**

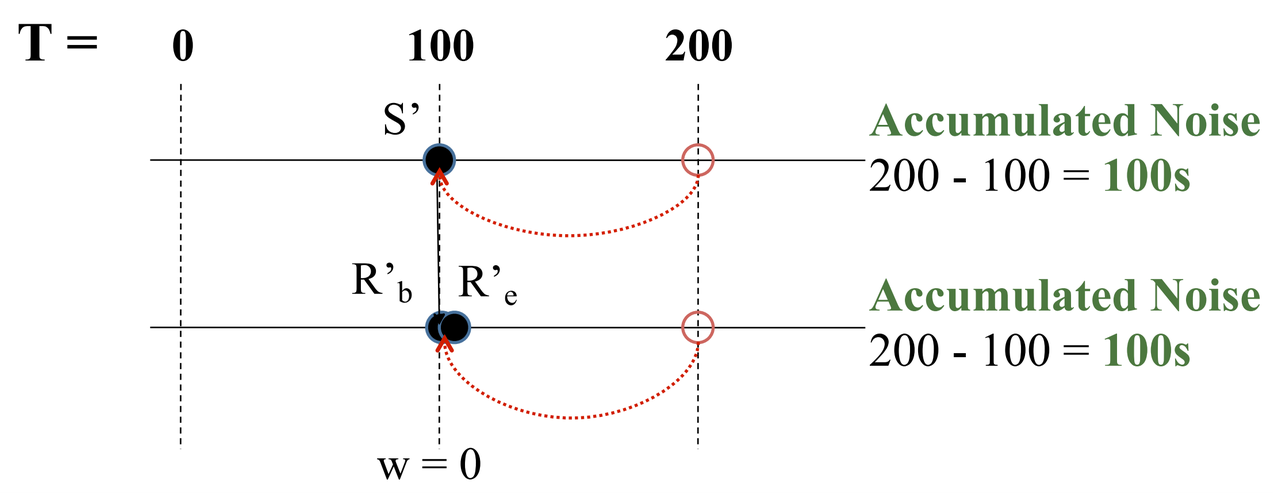



**Propagation - Rank 2 picks up 100s of delay even though its local-noise is 0!**

### Approximated Timeline

**Rank 1**
***Local* Noise = 100s**

**Rank 2**
***Local* Noise = 0s**



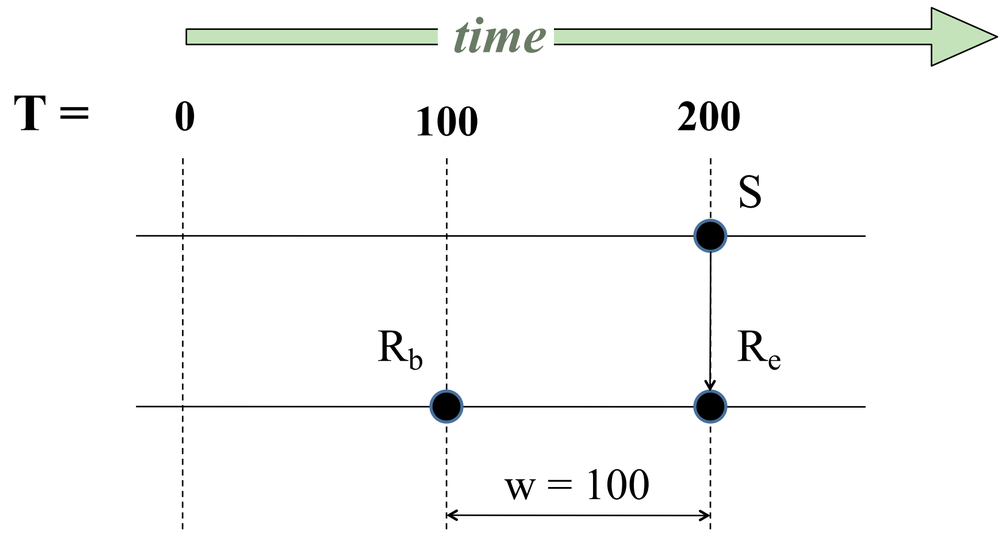
# Global Noise Estimation - Noise Absorption

Rank 1 Sends; Rank 2 Receives

Measured Timeline

**Rank 1**
***Local* Noise = 50s**

**Rank 2**
***Local* Noise = 100s**



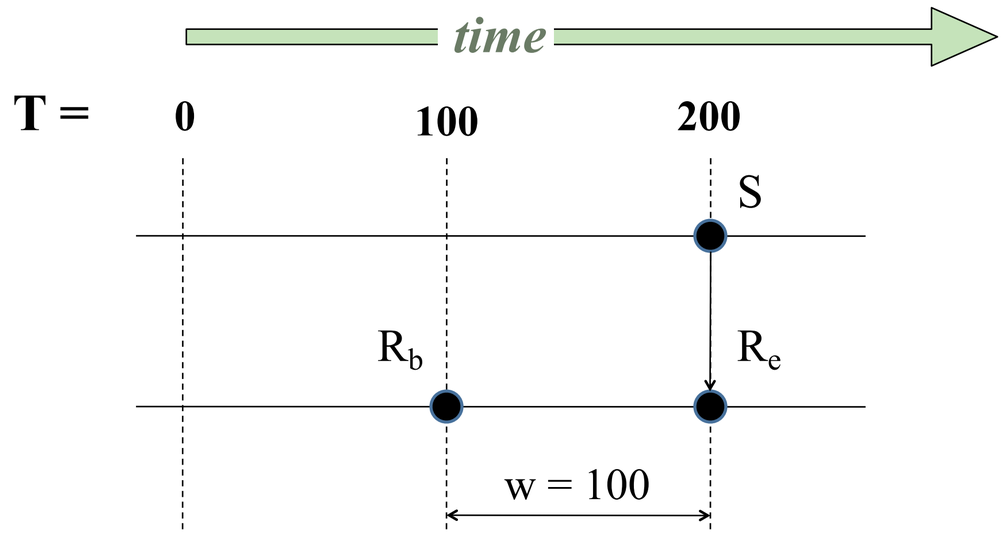
# Global Noise Estimation - Noise Absorption

## Rank 1 Sends; Rank 2 Receives

### Measured Timeline

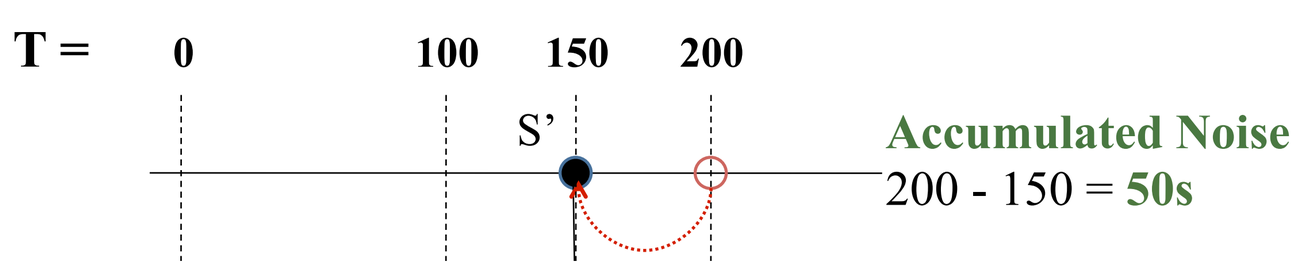
**Rank 1**
***Local* Noise = 50s**

**Rank 2**
***Local* Noise = 100s**




### Approximated Timeline

**Rank 1**
***Local* Noise = 50s**



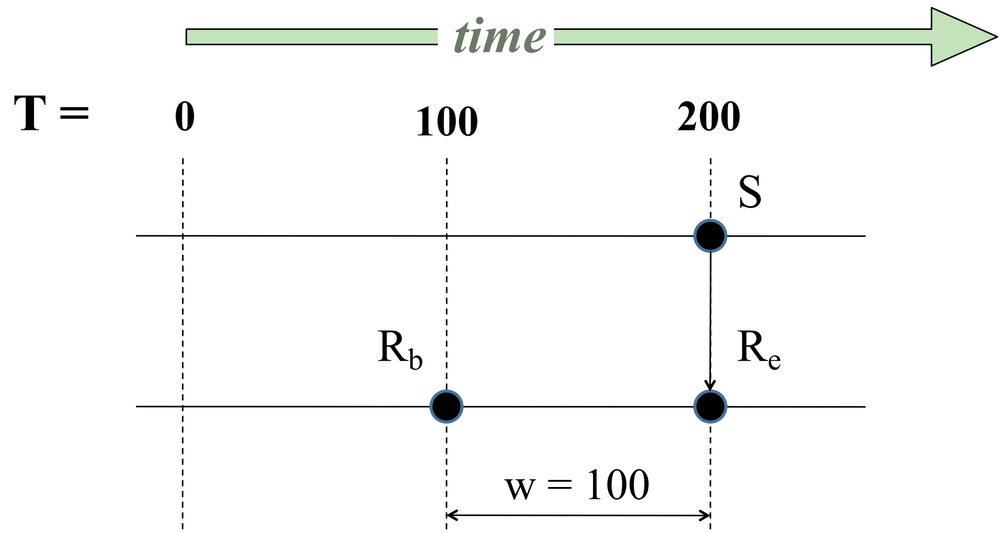
# Global Noise Estimation - Noise Absorption

## Rank 1 Sends; Rank 2 Receives

### Measured Timeline

**Rank 1**
***Local* Noise = 50s**

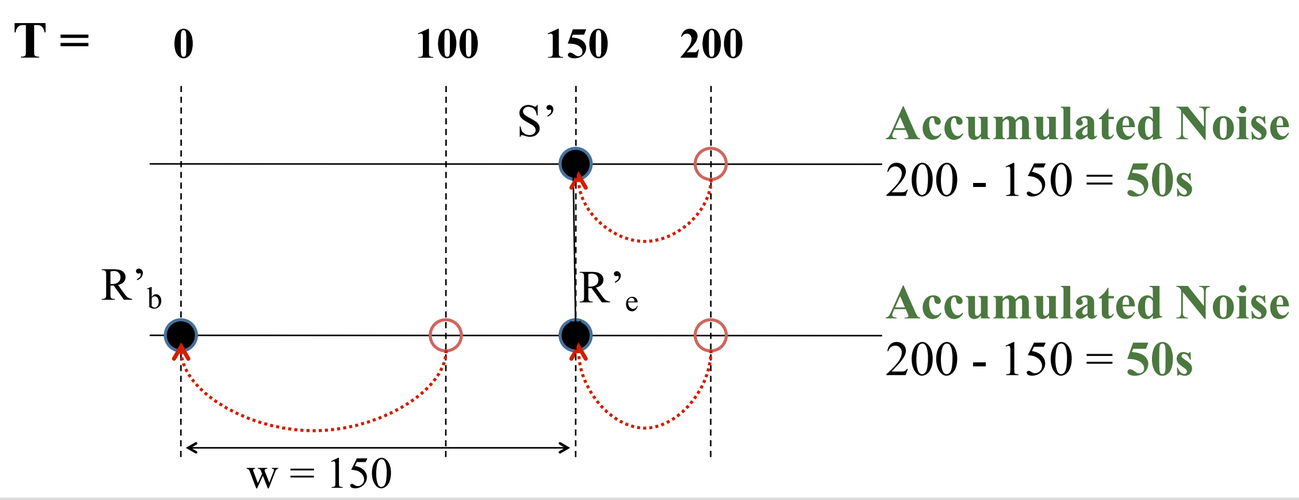
**Rank 2**
***Local* Noise = 100s**




### Approximated Timeline

**Rank 1**
***Local* Noise = 50s**

**Rank 2**
***Local* Noise = 100s**



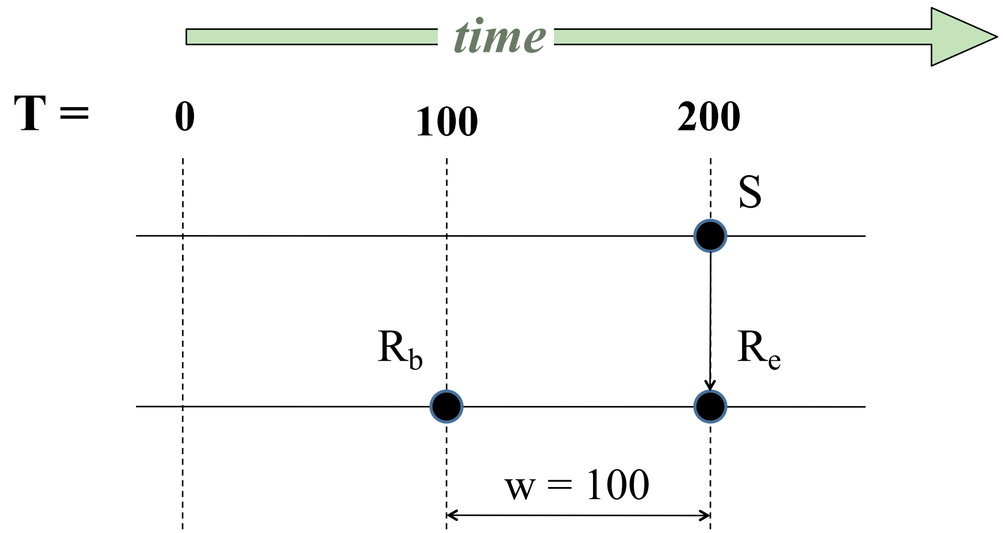
# Global Noise Estimation - Noise Absorption

Rank 1 Sends; Rank 2 Receives

Measured Timeline

**Rank 1**
***Local* Noise = 50s**

**Rank 2**
***Local* Noise = 100s**

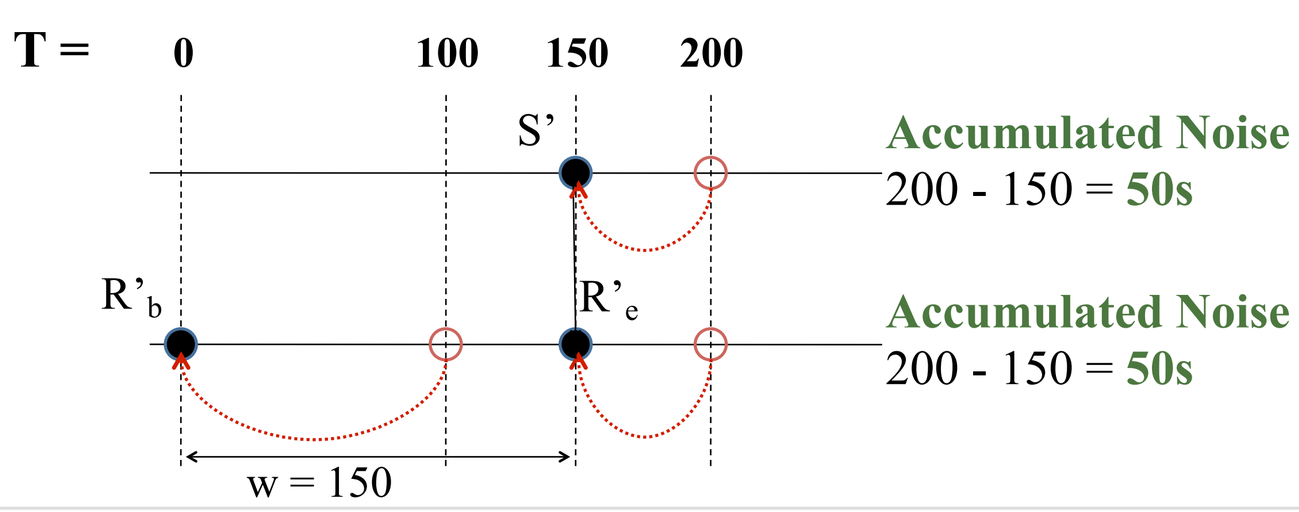



**Absorption - Rank 2's Acc. Noise is only 50s, but local noise was 100s. It loses 50s!**

Approximated Timeline

**Rank 1**
***Local* Noise = 50s**

**Rank 2**
***Local* Noise = 100s**



# *Prior Work on Trace-based Timeline Approximation*

- Wolf, Malony, Shende and Morris | HPCC'06
  - Context: Measurement perturbation compensation.
- Sottile, Chandu and Bader | IPDPS'06
  - Trace-based simulation
  - Inject artificial noise into existing application traces and adjust timestamps
  - Reverse of our current work - we remove existing, real noise effects from trace.

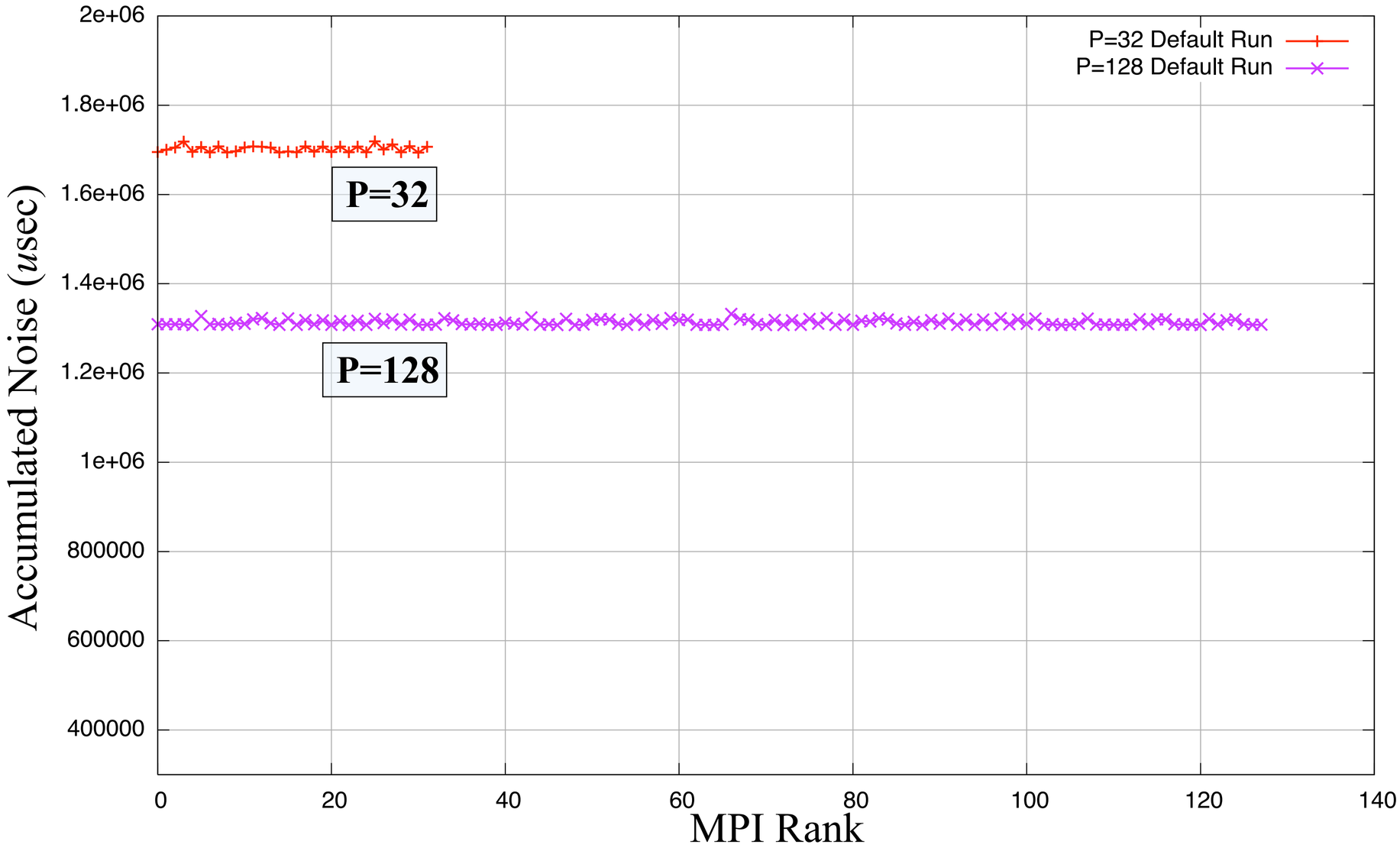
# *Noise-Effect Estimation Demonstrated*

- Demonstrate estimation of delay caused in *application/benchmark* by *real noise*
- Platform: 32 (2x2) Opterons (P=128); GigE; Linux w/ KTAU; SDSC
- Application : Sweep 3D (kernel representative of ASC applications)
  - Repeating phases of Send/Recv followed by Allreduce
  - Problem - 650^3, 15 iterations, MPI based
- Scaling
  - Strong
  - P=32, P=128
- Instrumentation / Measurement
  - TAU Tracing of MPI events
  - Associating KTAU OS Metrics with the events
- Analysis
  - Run Trace analysis (described earlier) to calculate delay due to noise

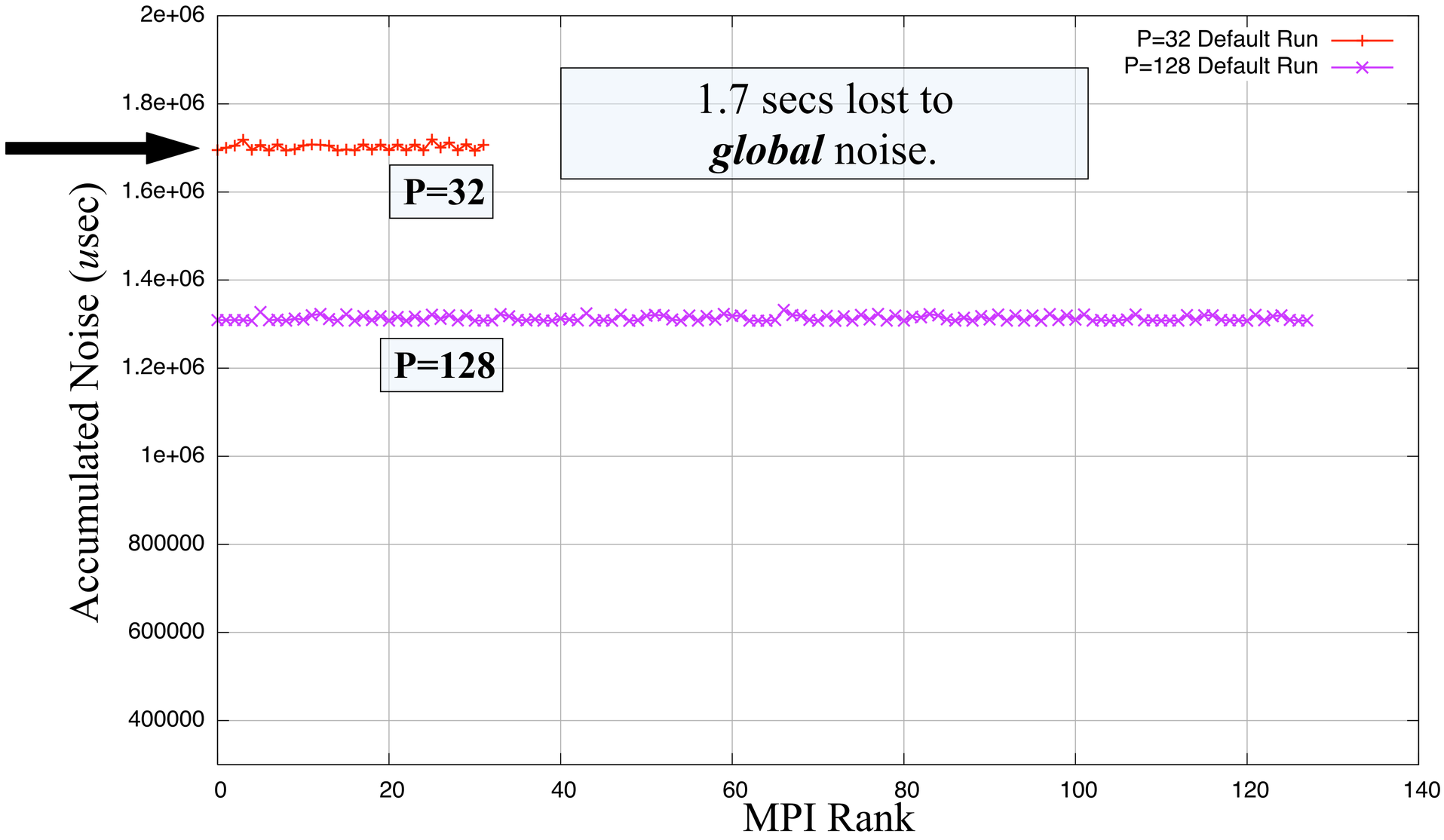
# *Noise Sources and Metrics*

- OS Noise Sources
  - global timer interrupt - keeps time & timers, intervals (10, 4, 1 msec)
  - local timer interrupt - update process times (scheduling), every cpu/core
  - preemptive schedule - duration preempted
- Metrics
  - Accumulated Noise
    - Estimate of delay due to global noise effects, in secs
    - By how much time would the application have run faster w/o noise?
  - Noise Amplification Ratio
    - Accumulated Noise / Local Noise
    - How much was noise amplified? How much was absorbed?
  - Lesser means better for both metrics
- Metrics calculated seperately for each ***noise source*** and ***combined noise***

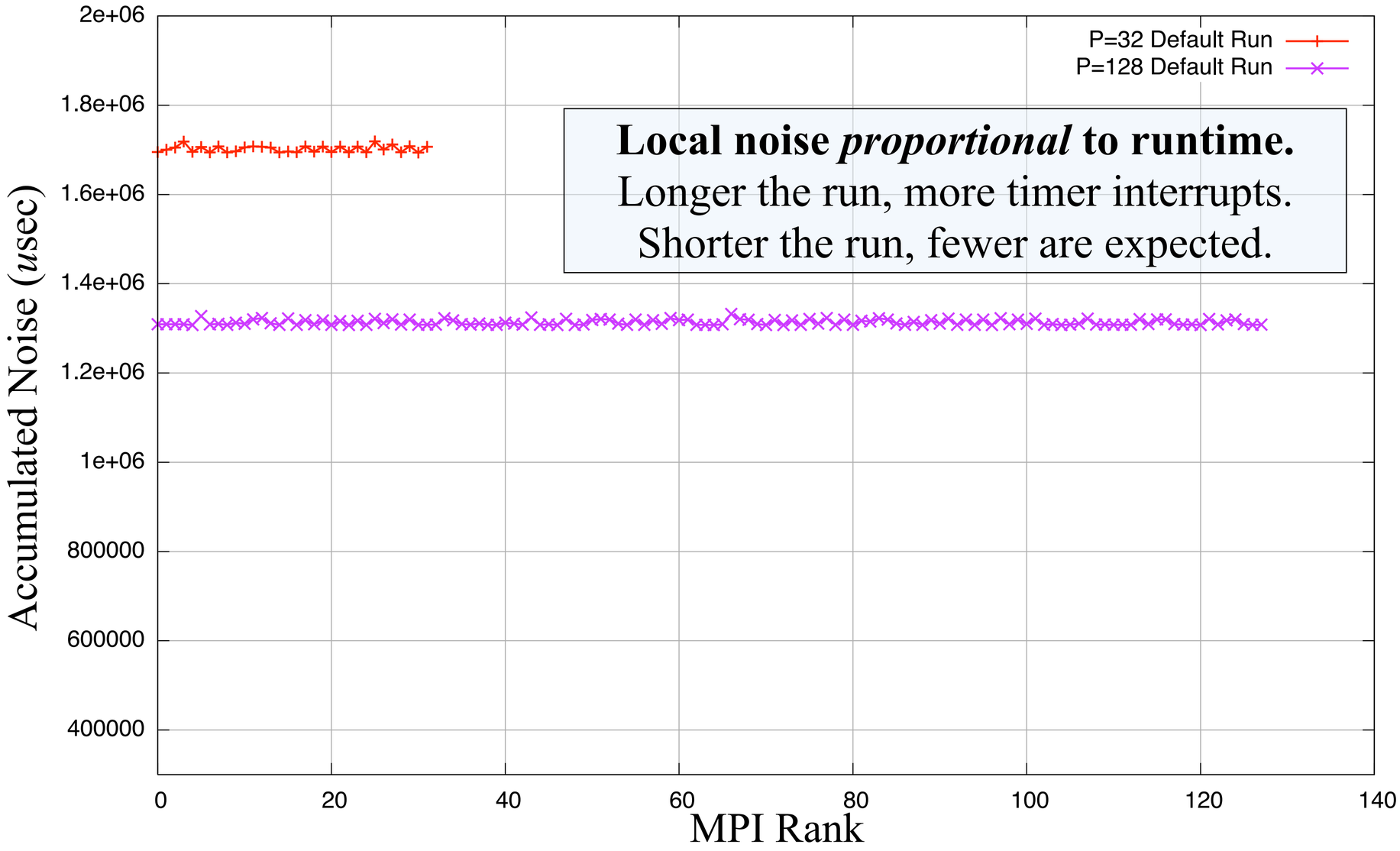
# Overall Accumulated Noise - Effect of Scaling



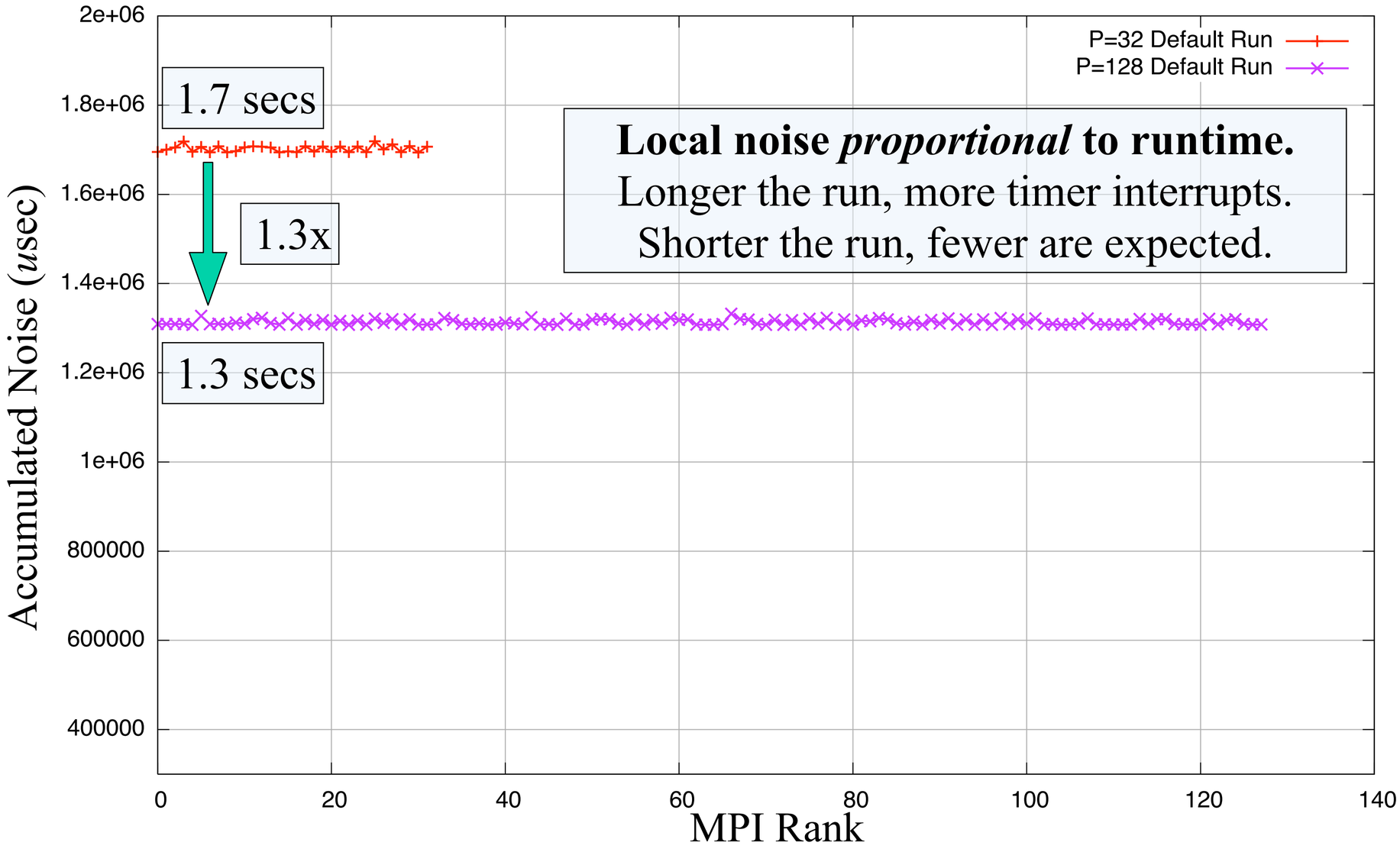
# Overall Accumulated Noise - Effect of Scaling



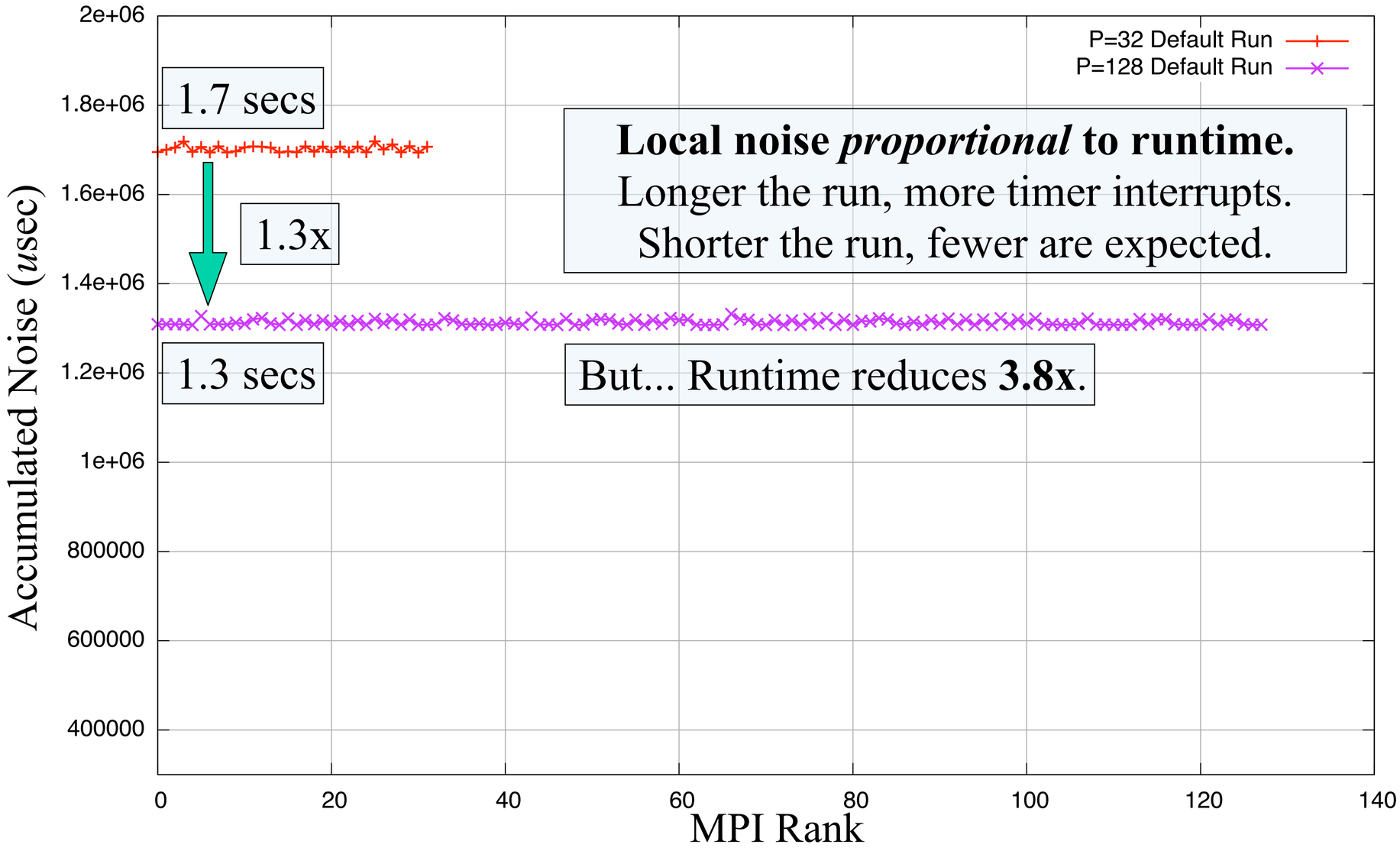
# *Overall Accumulated Noise - Effect of Scaling*



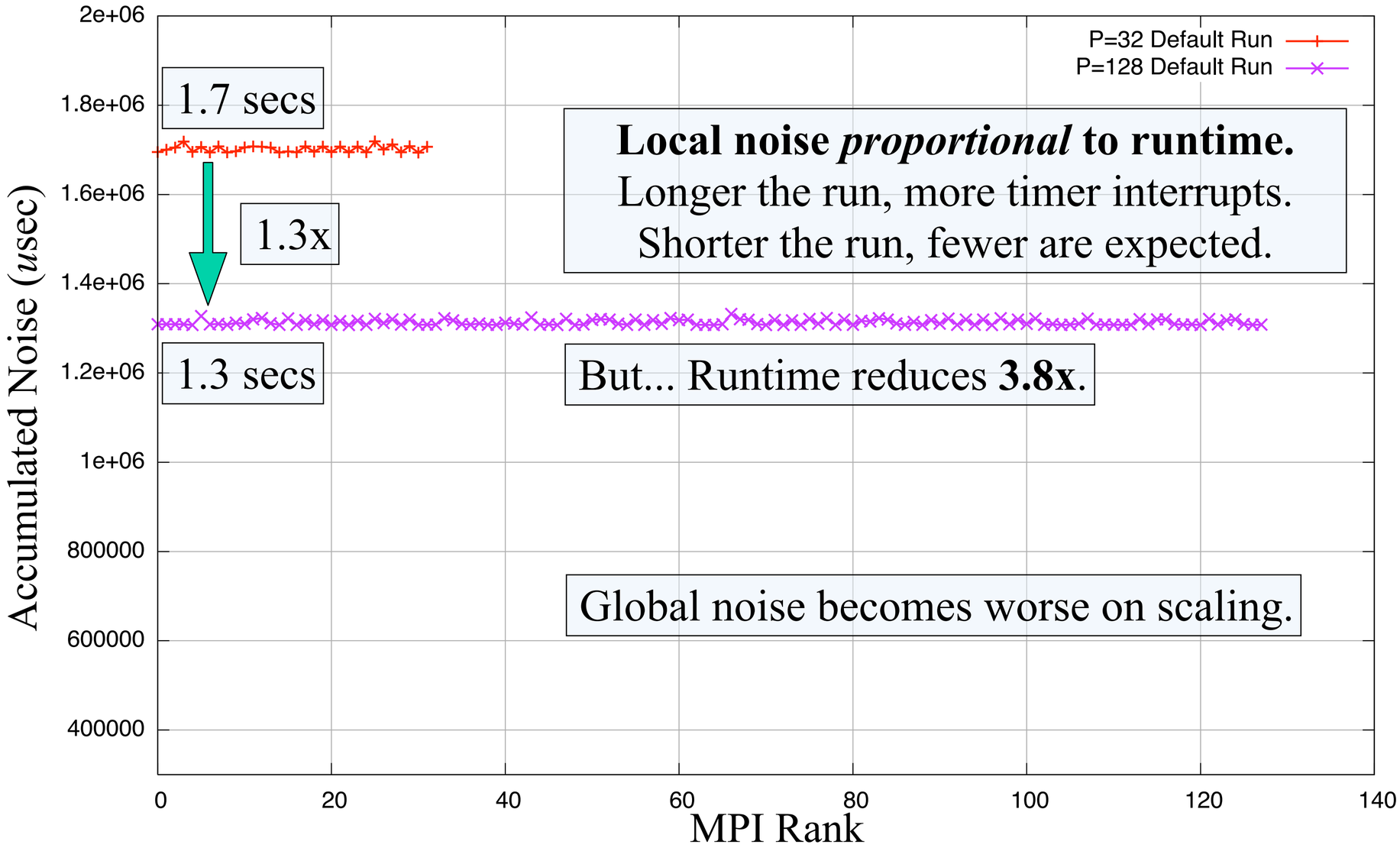
# Overall Accumulated Noise - Effect of Scaling



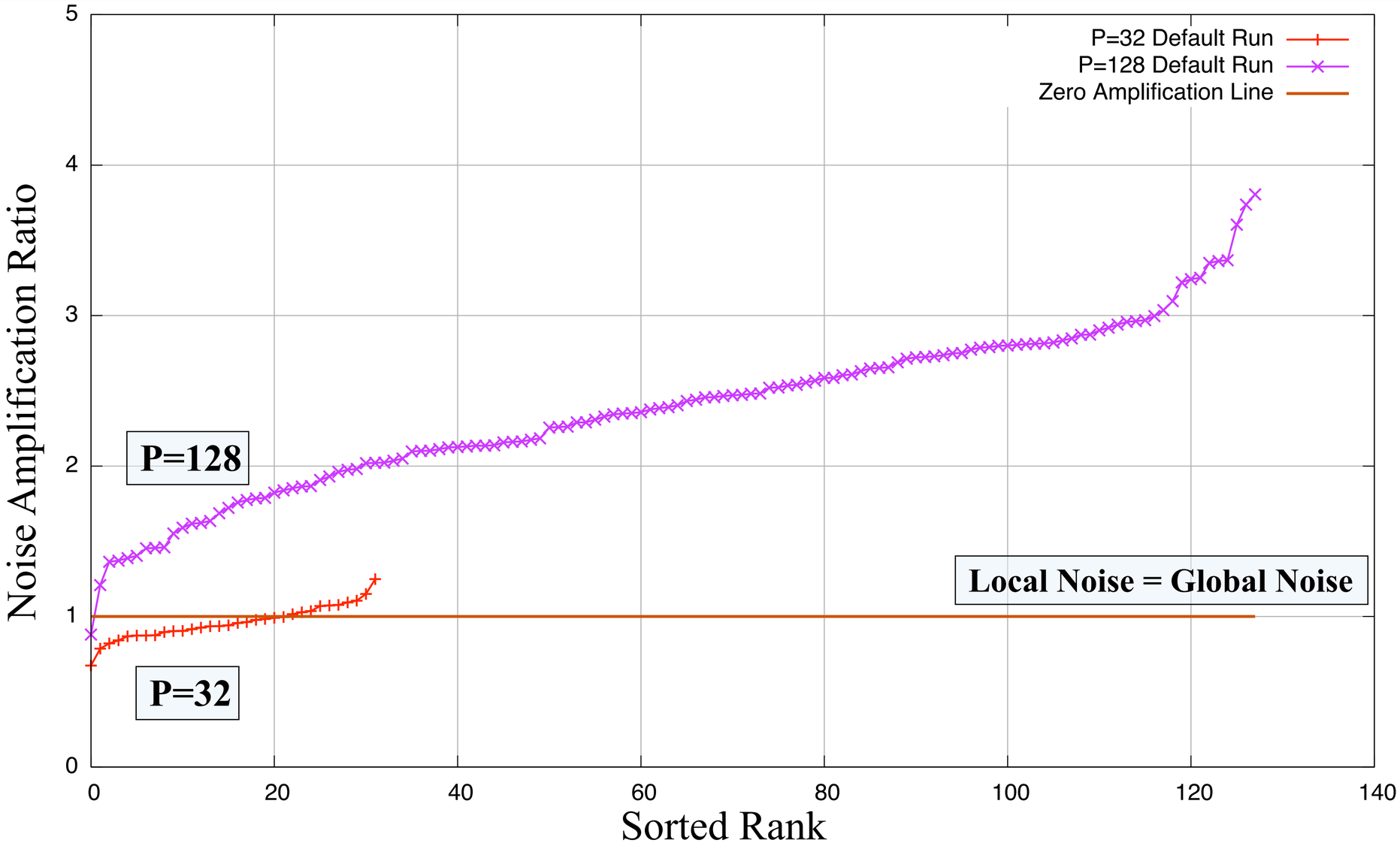
# *Overall Accumulated Noise - Effect of Scaling*



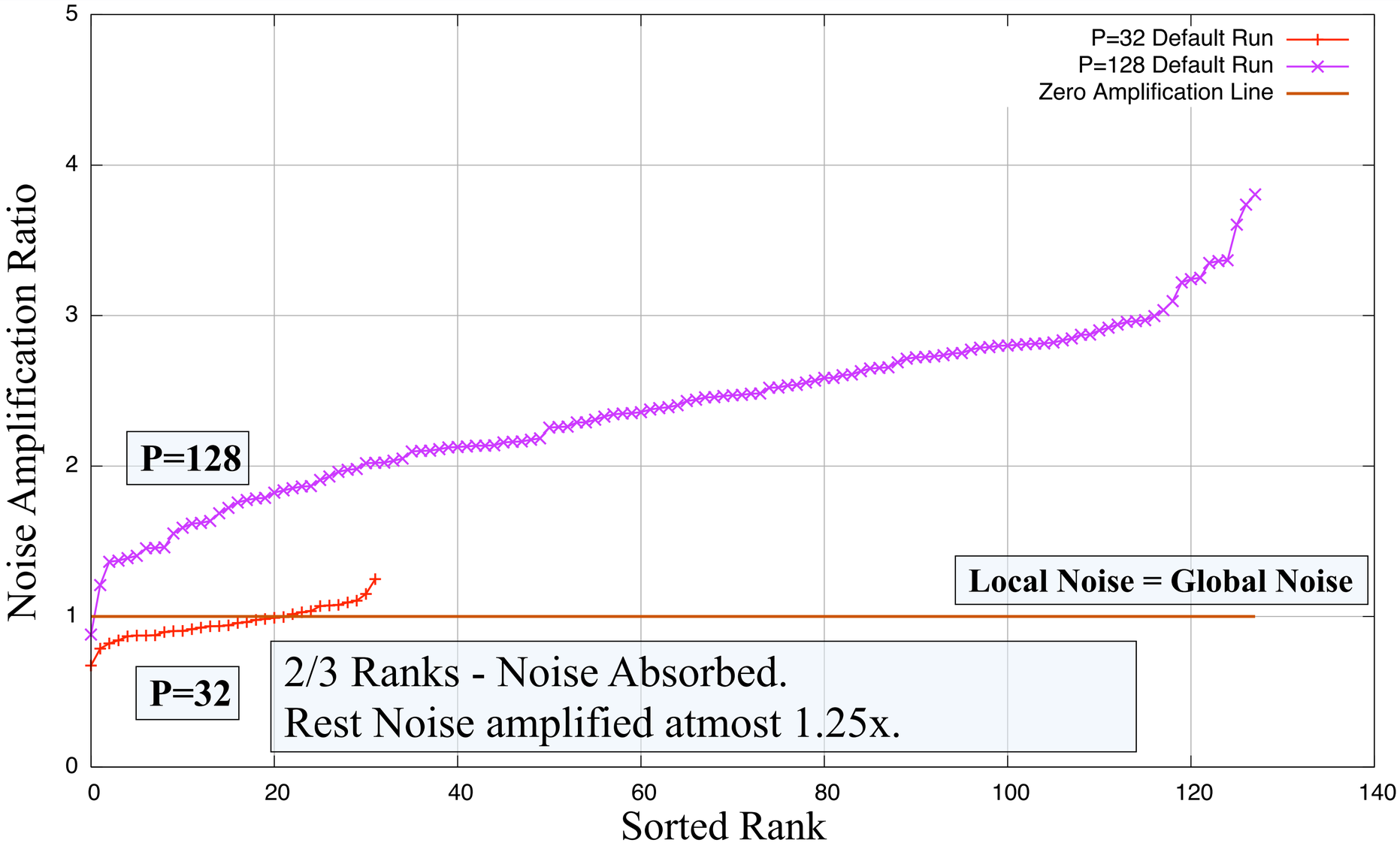
# *Overall Accumulated Noise - Effect of Scaling*



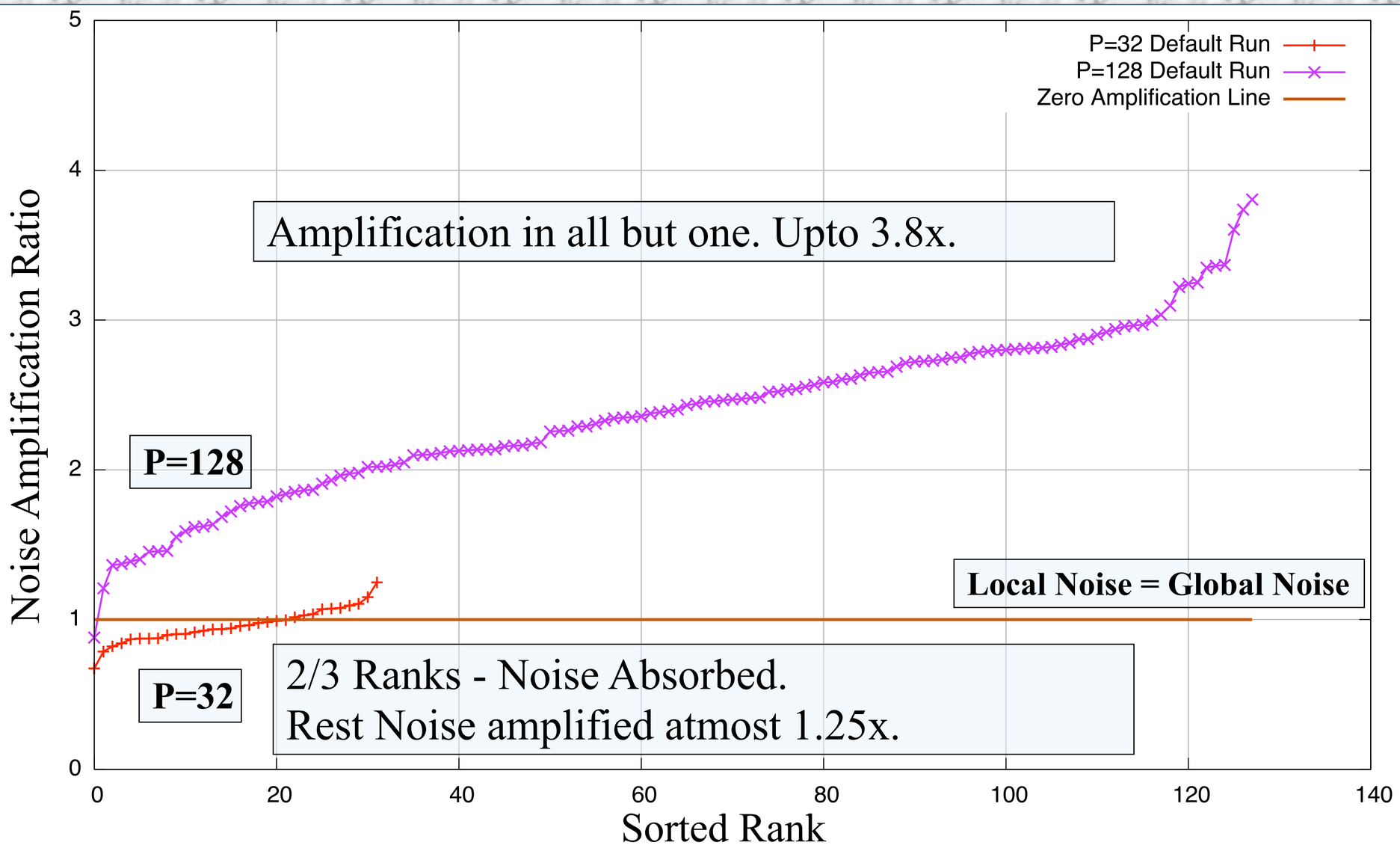
# Overall Noise Amplification Ratio - Effect of Scaling



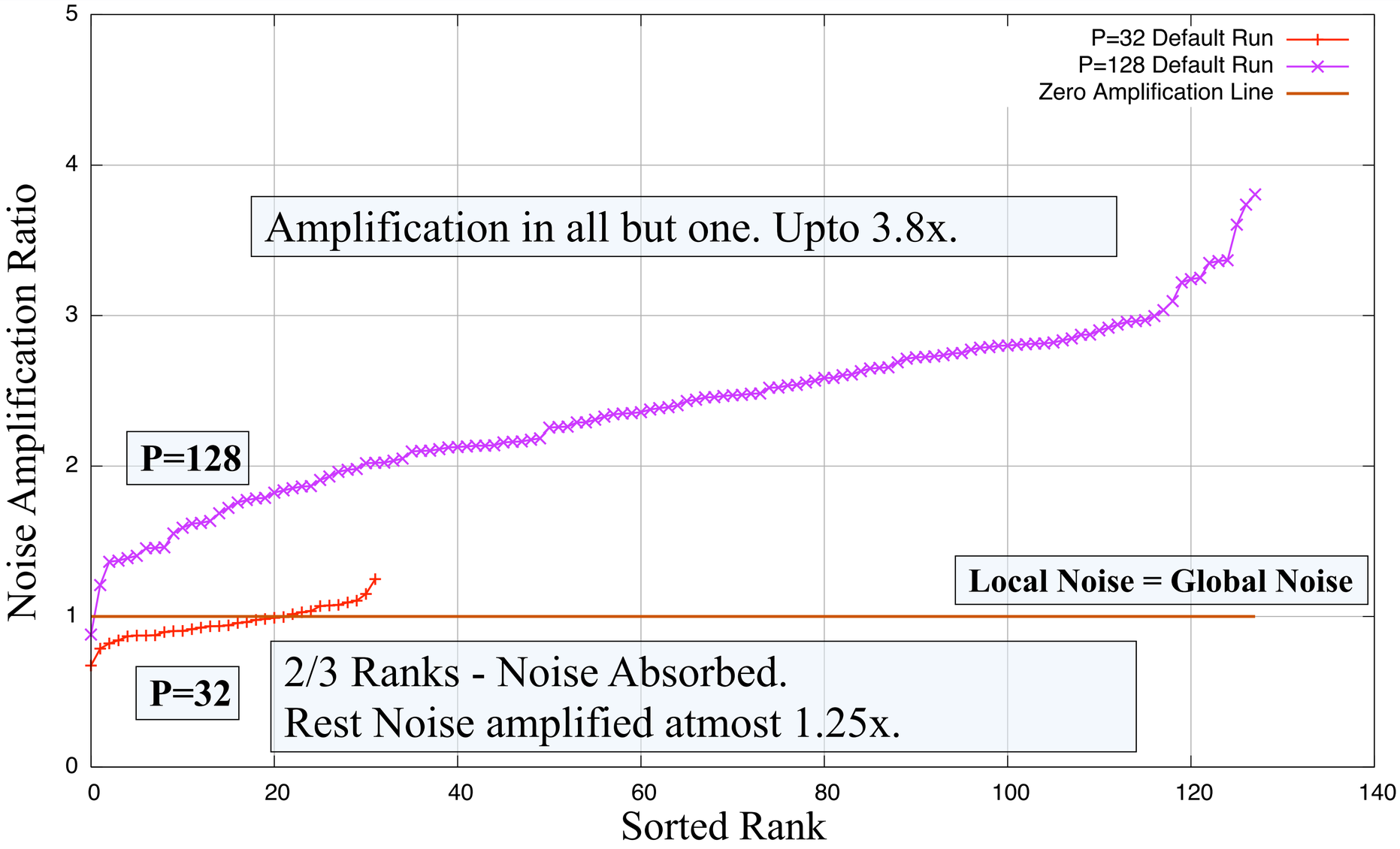
# Overall Noise Amplification Ratio - Effect of Scaling



# Overall Noise Amplification Ratio - Effect of Scaling

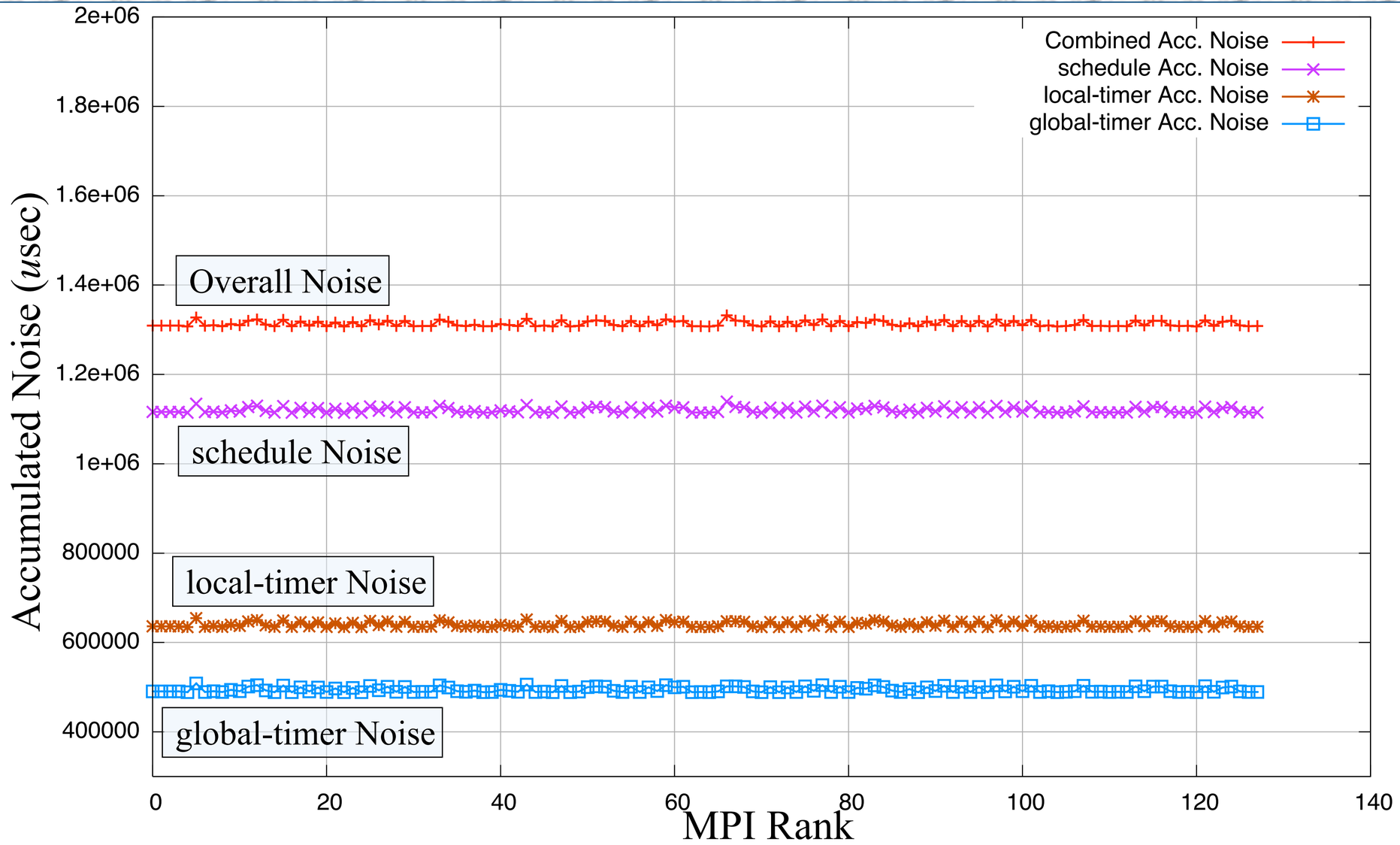


# Overall Noise Amplification Ratio - Effect of Scaling

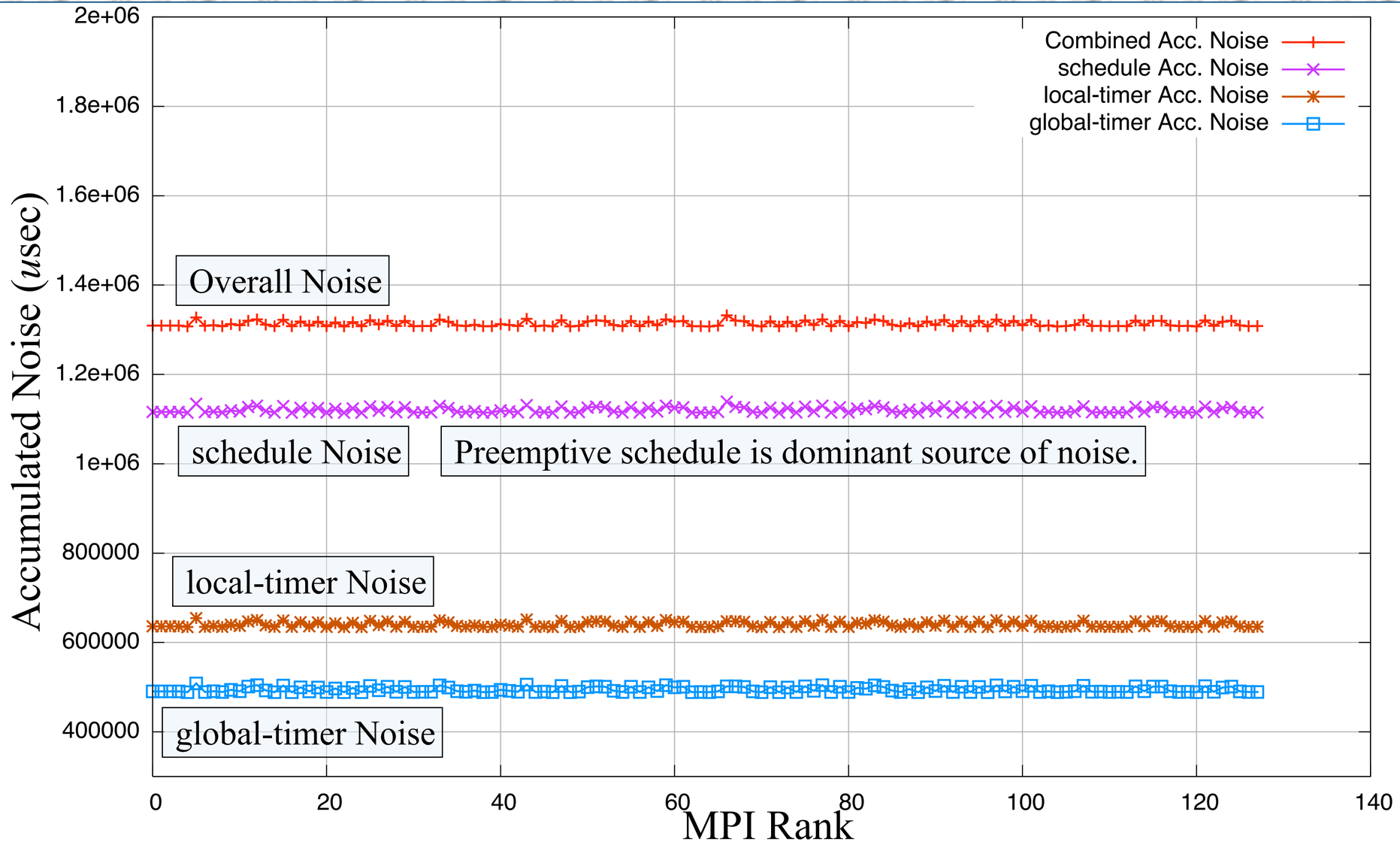



**Confirms that scaling is increasing global noise-effect.**
**How do the noise sources contribute?**

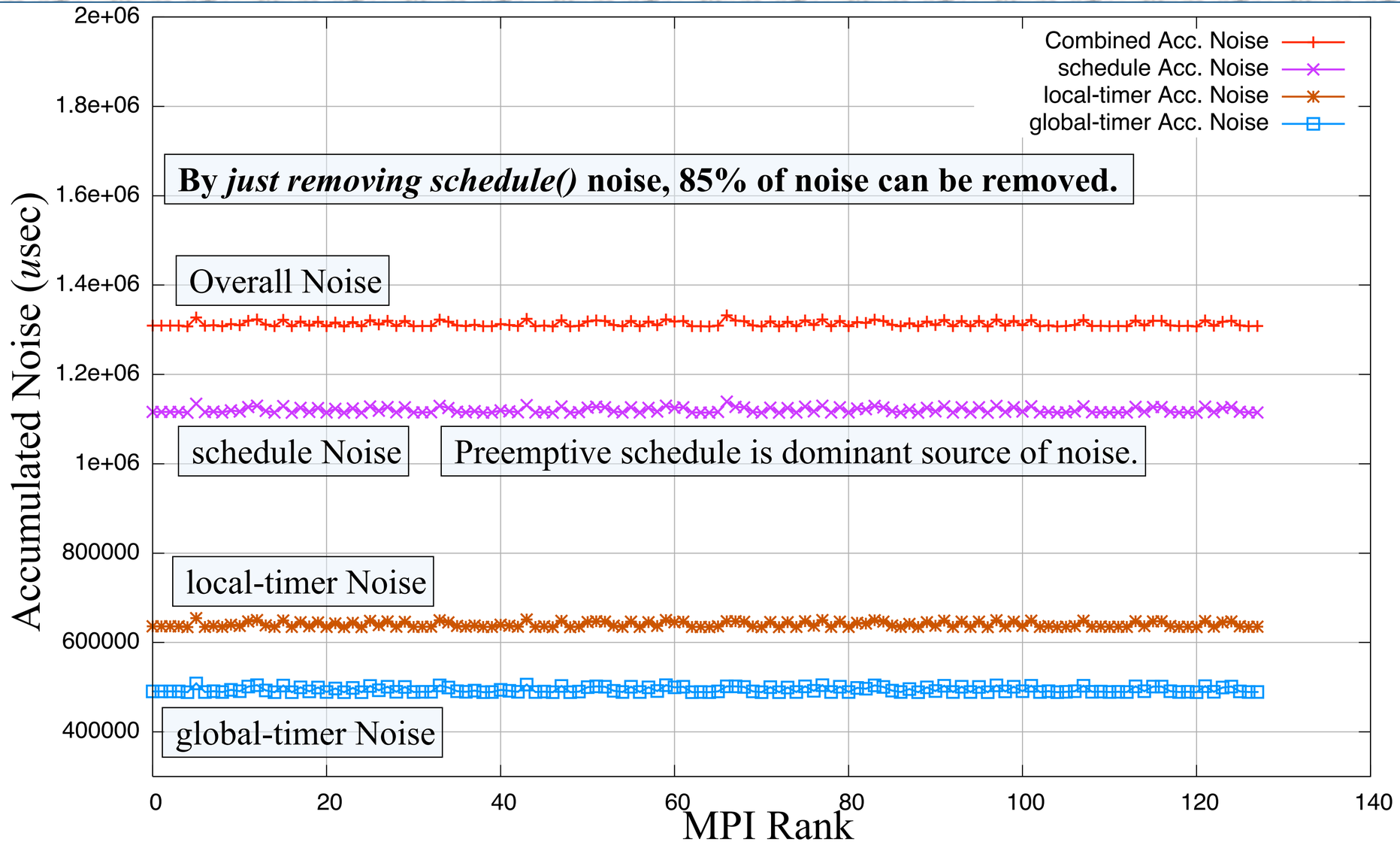
# *Noise Sources -- Accumulated Noise -- P=128*



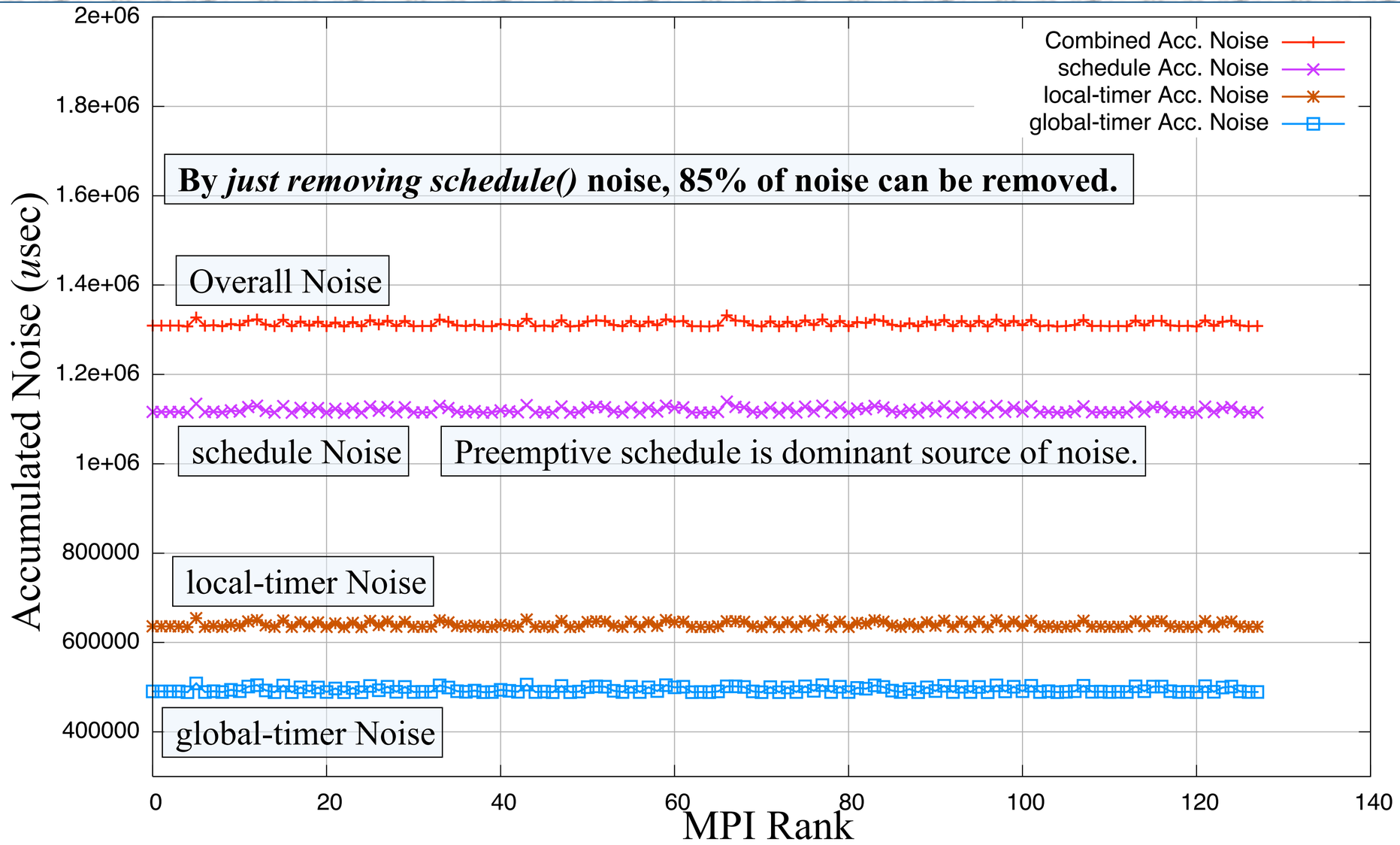
# Noise Sources -- Accumulated Noise -- P=128



# *Noise Sources -- Accumulated Noise -- P=128*

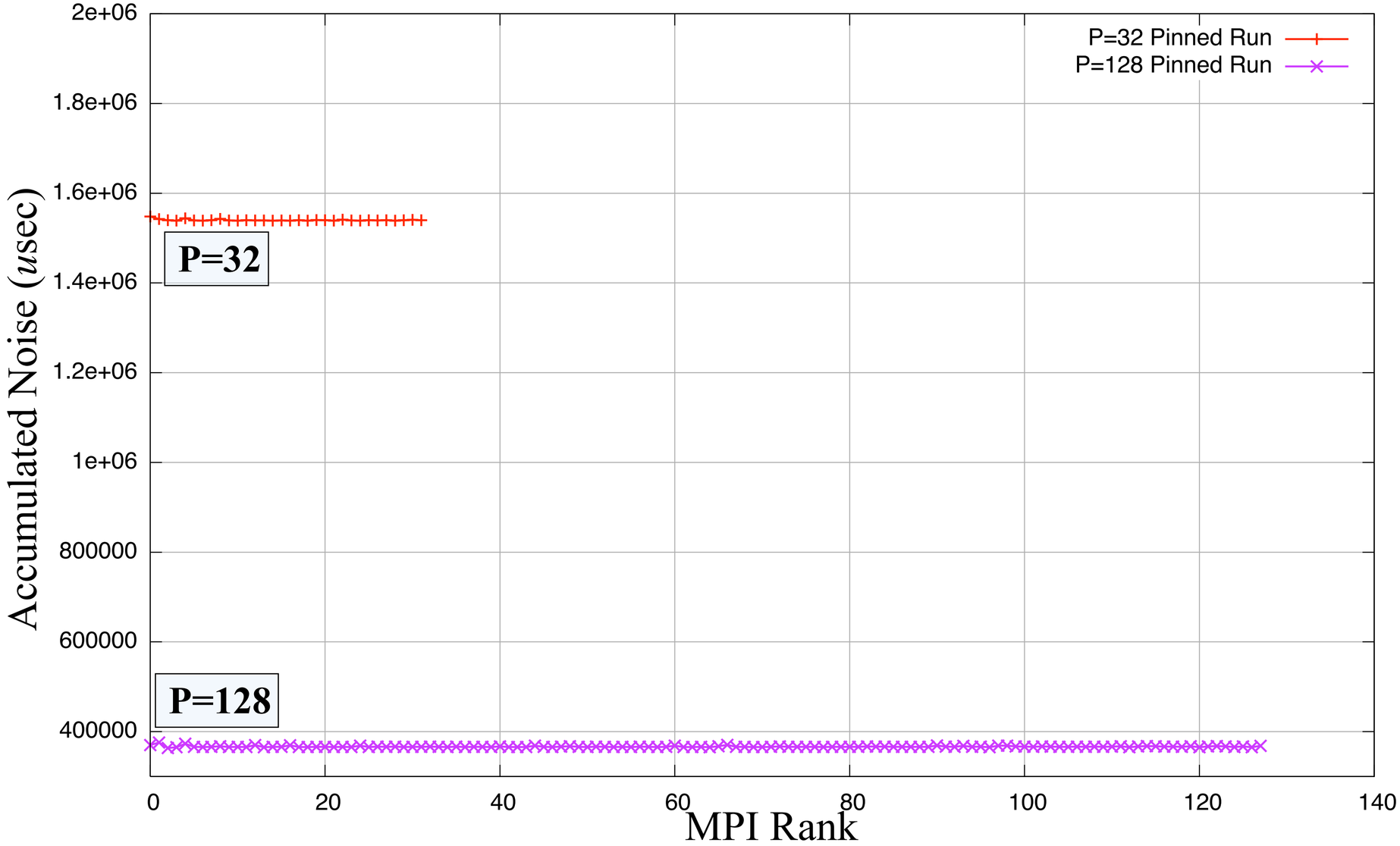


# *Noise Sources -- Accumulated Noise -- P=128*

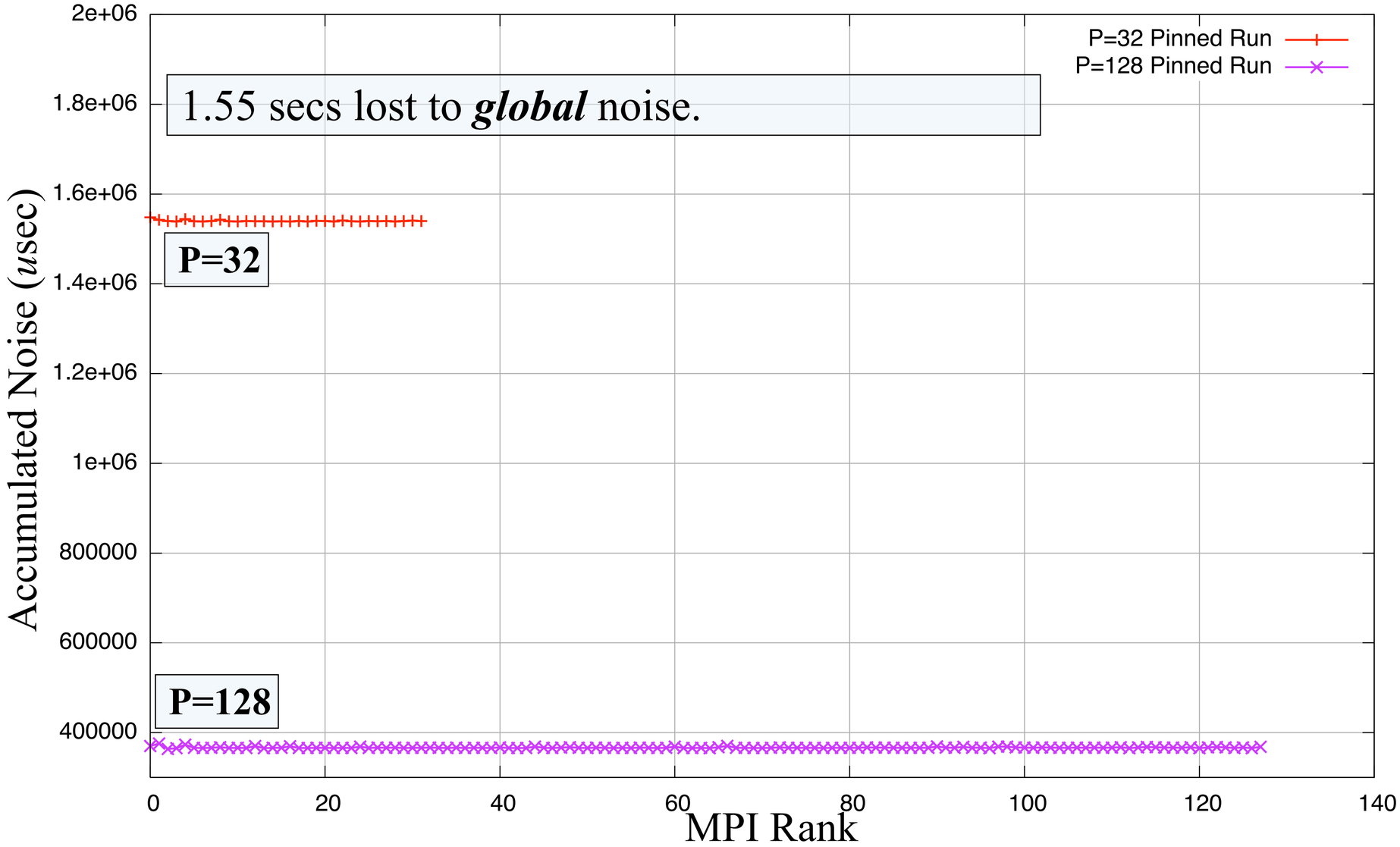



Lets make a simple change that affects scheduling.
***Pin* the ranks to the processors.**
***What happens?***

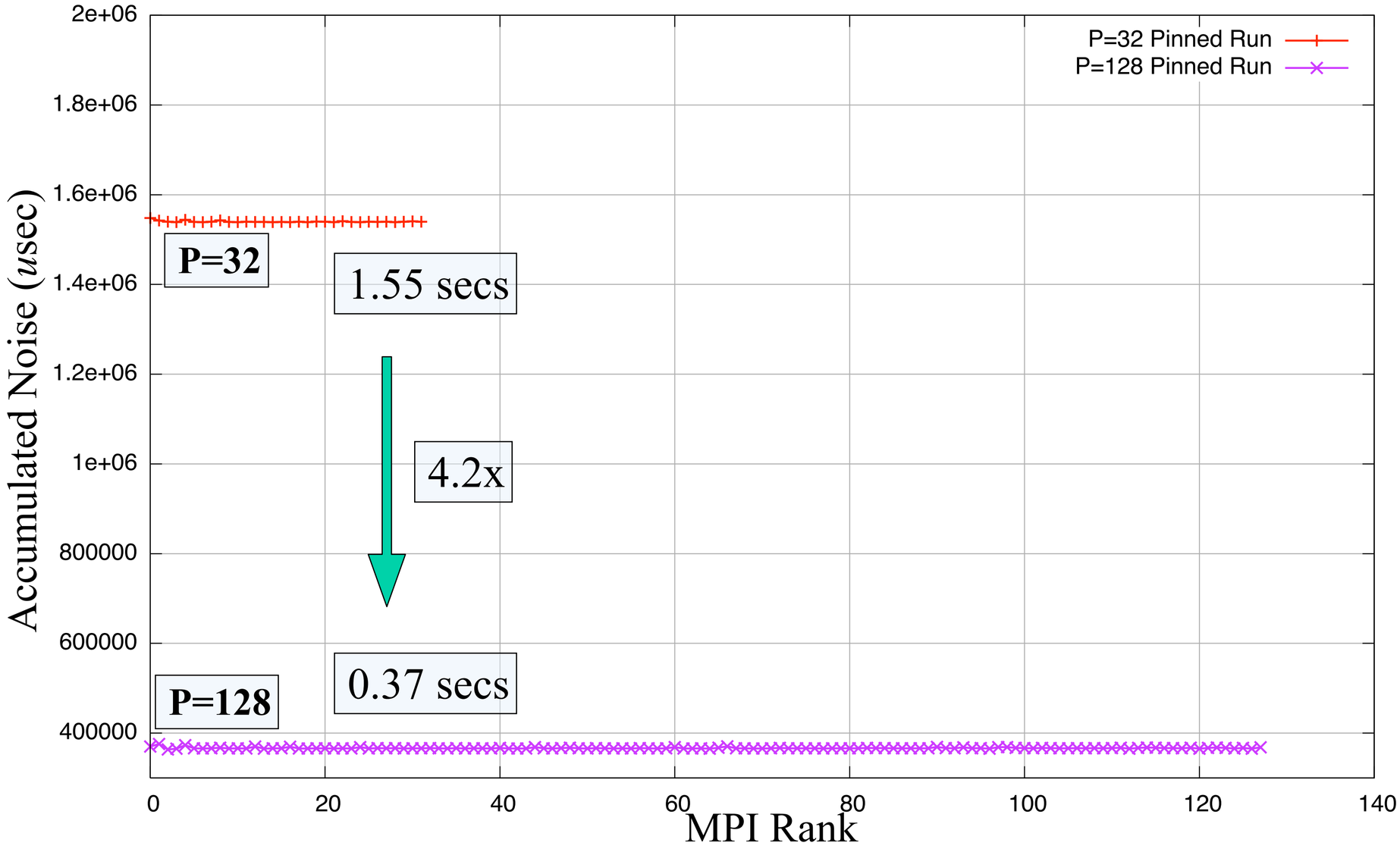
# Overall Accumulated Noise - Pinned - Effect of Strong Scaling



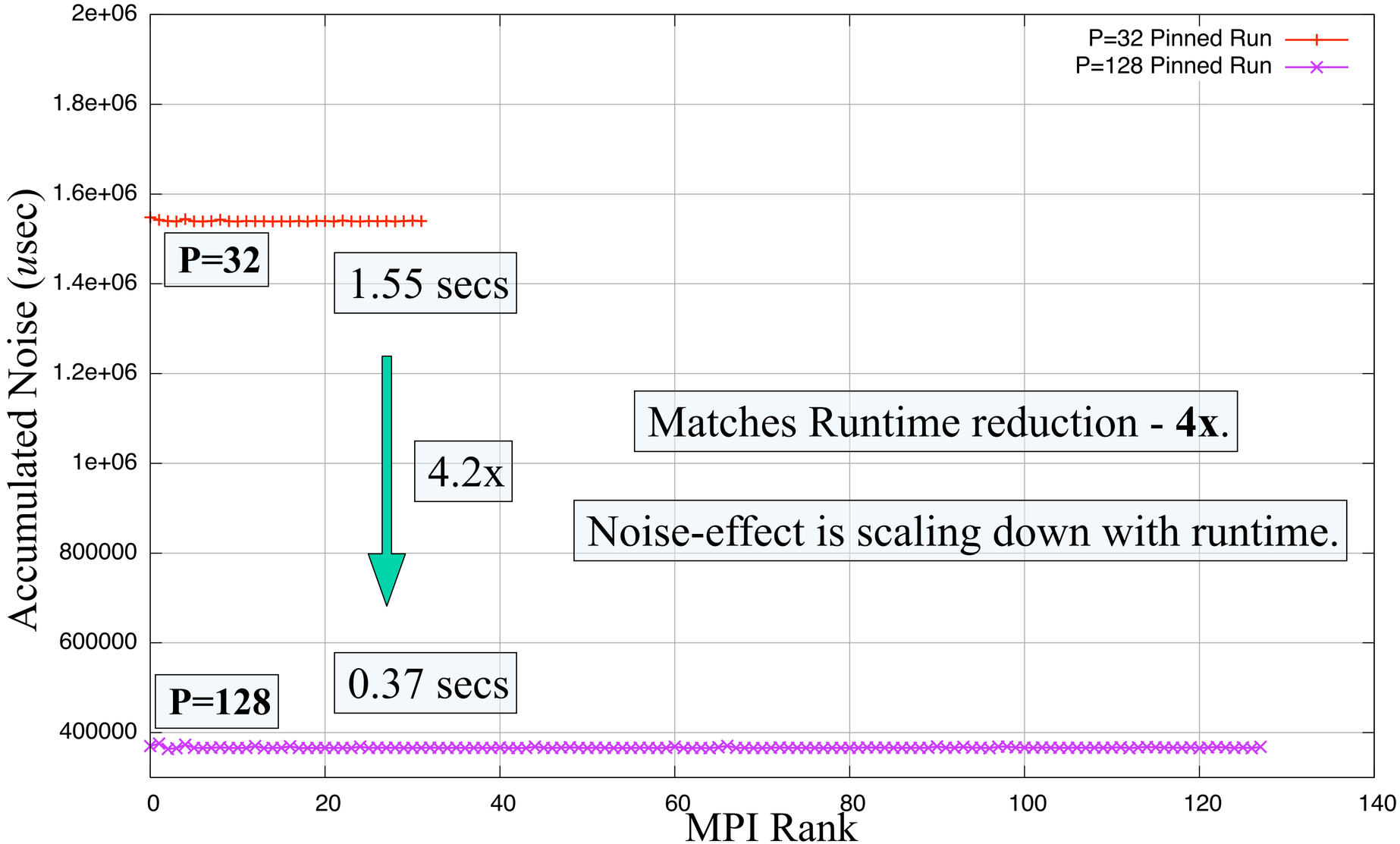
# Overall Accumulated Noise - Pinned - Effect of Strong Scaling



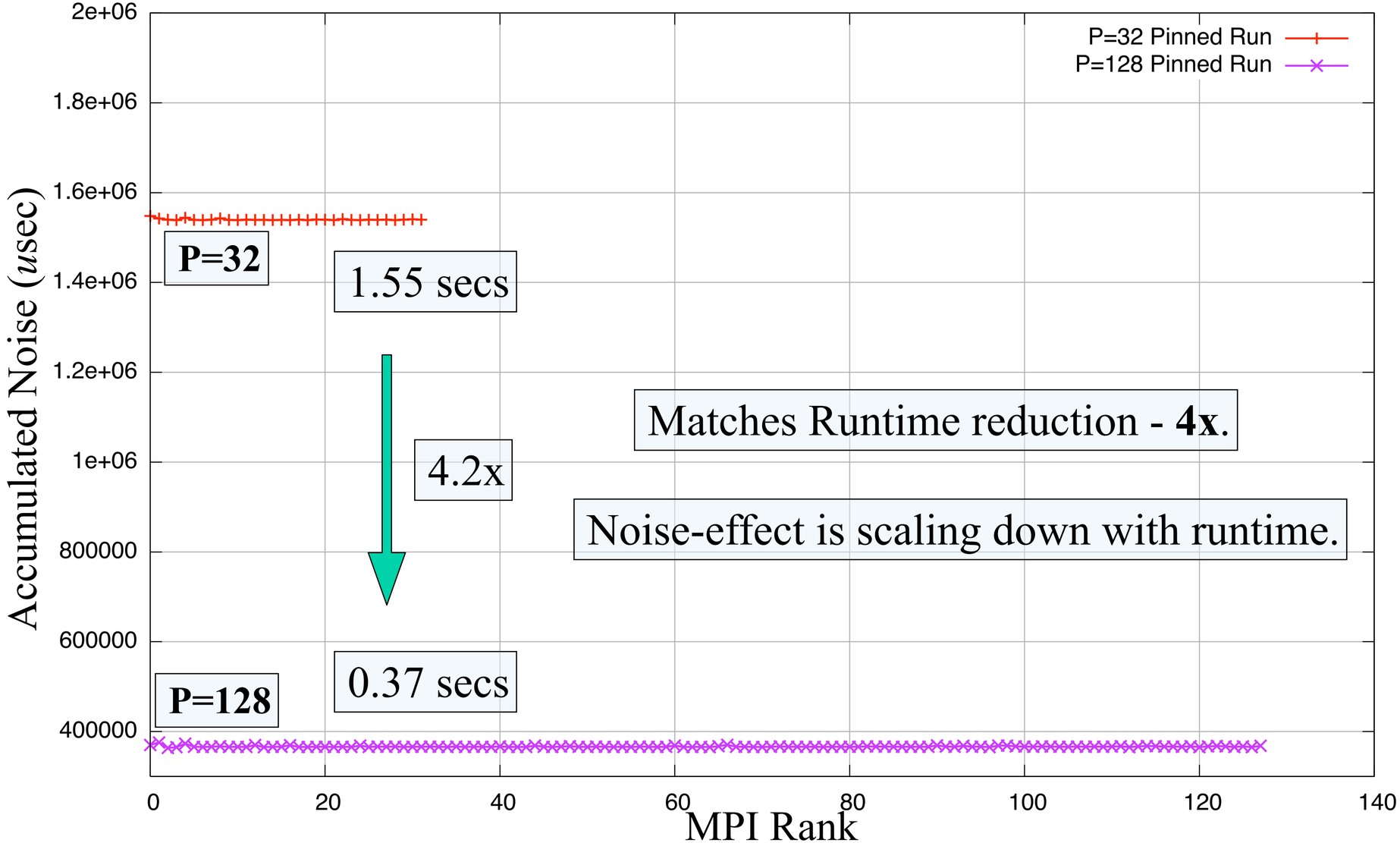
# Overall Accumulated Noise - Pinned - Effect of Strong Scaling



# Overall Accumulated Noise - Pinned - Effect of Strong Scaling

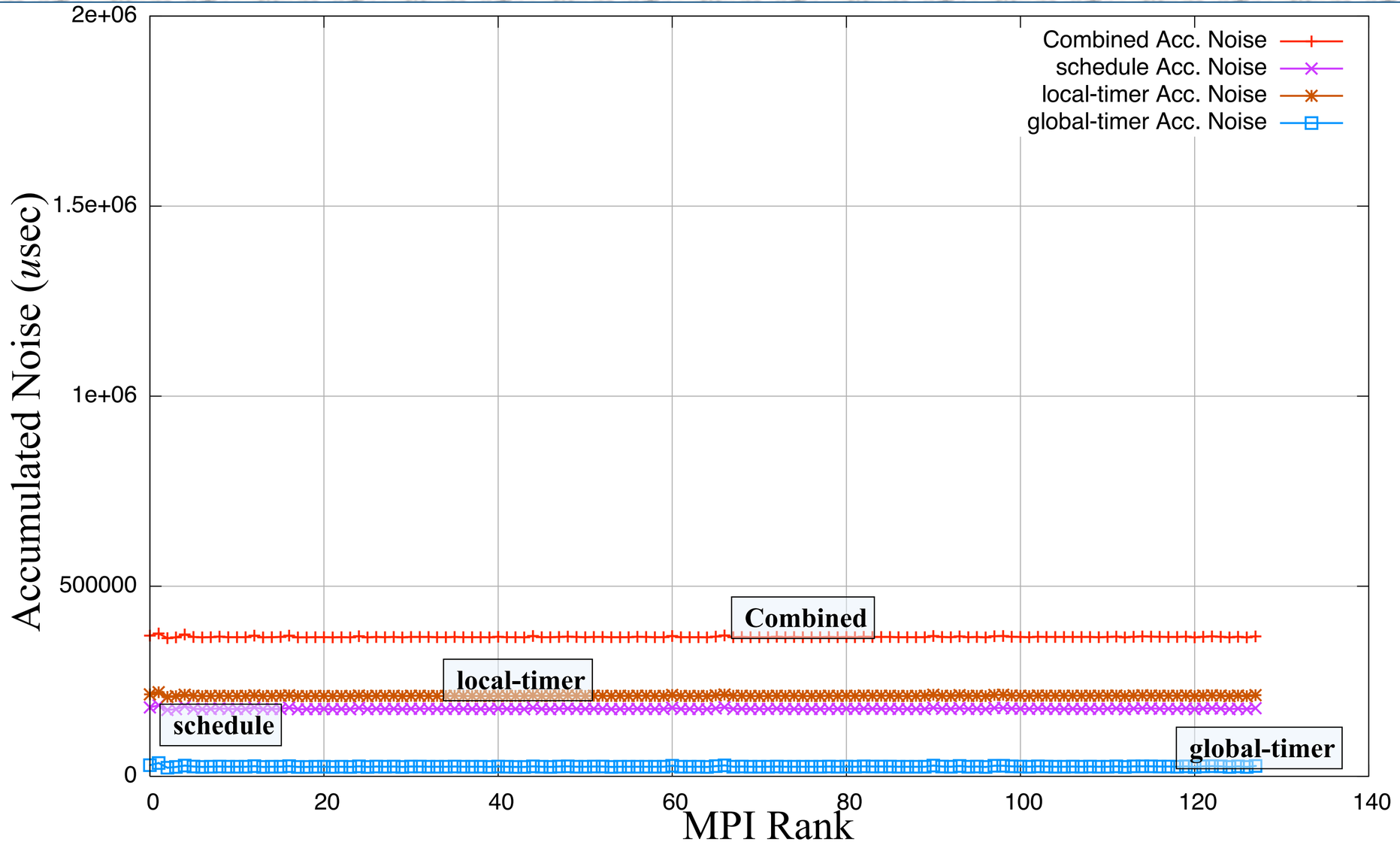


# *Overall Accumulated Noise - Pinned - Effect of Strong Scaling*

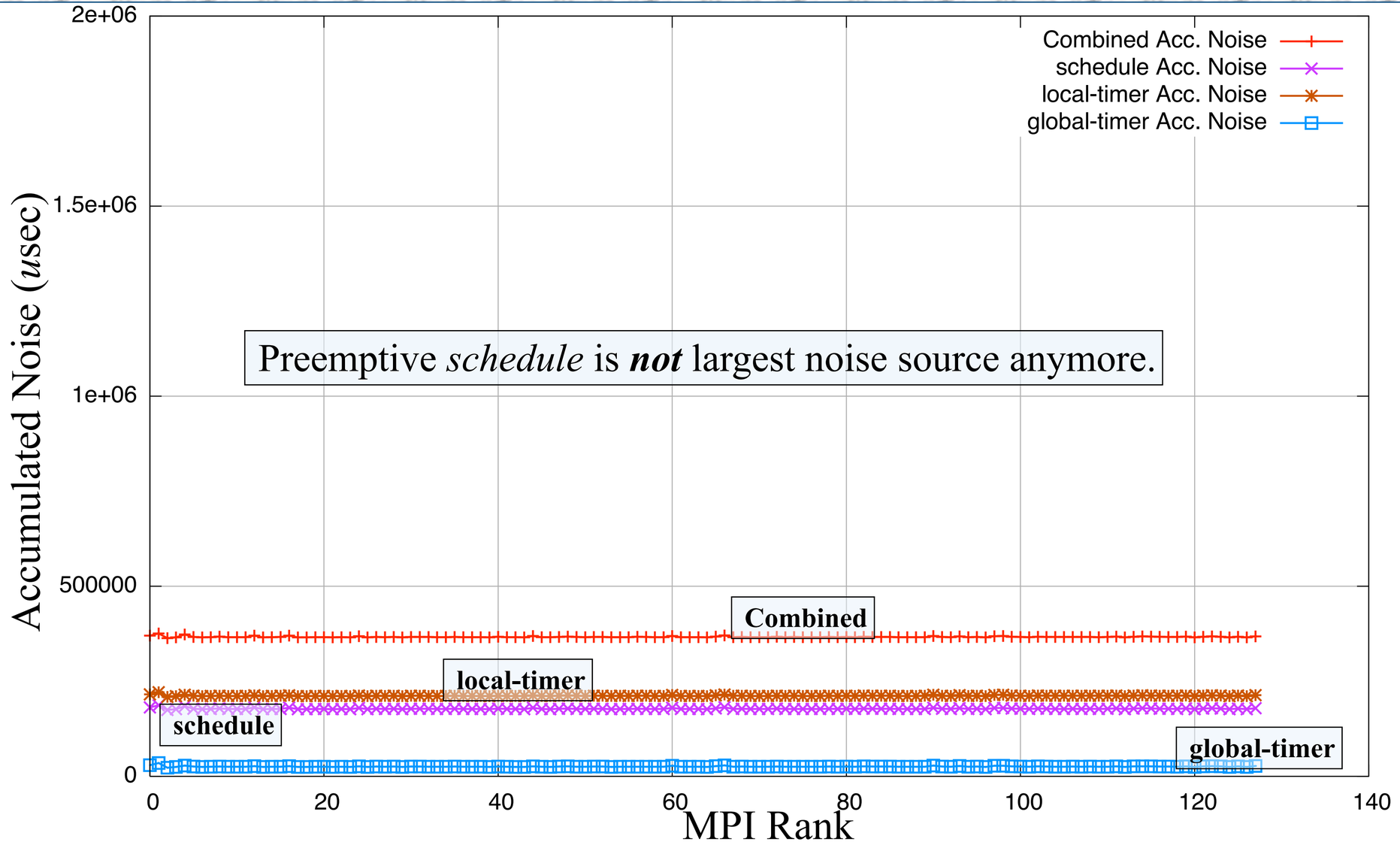



Lets look at the ***different noise sources*** again...

# The Noise Sources -- Pinned -- P=128



# The Noise Sources -- Pinned -- P=128



# *Small Noise Effects*

- Magnitude of the Accumulated Noise - 1.3 secs
  - Represents approx. 1% of runtime (132 secs)

- Small cluster (32 nodes) + Slow interconnect (GigE)
  - => Small global-noise effect

- For large OS noise related slowdowns
  - Larger scales
  - Fast interconnect

- Global-Noise Estimation still detected and revealed interesting noise features

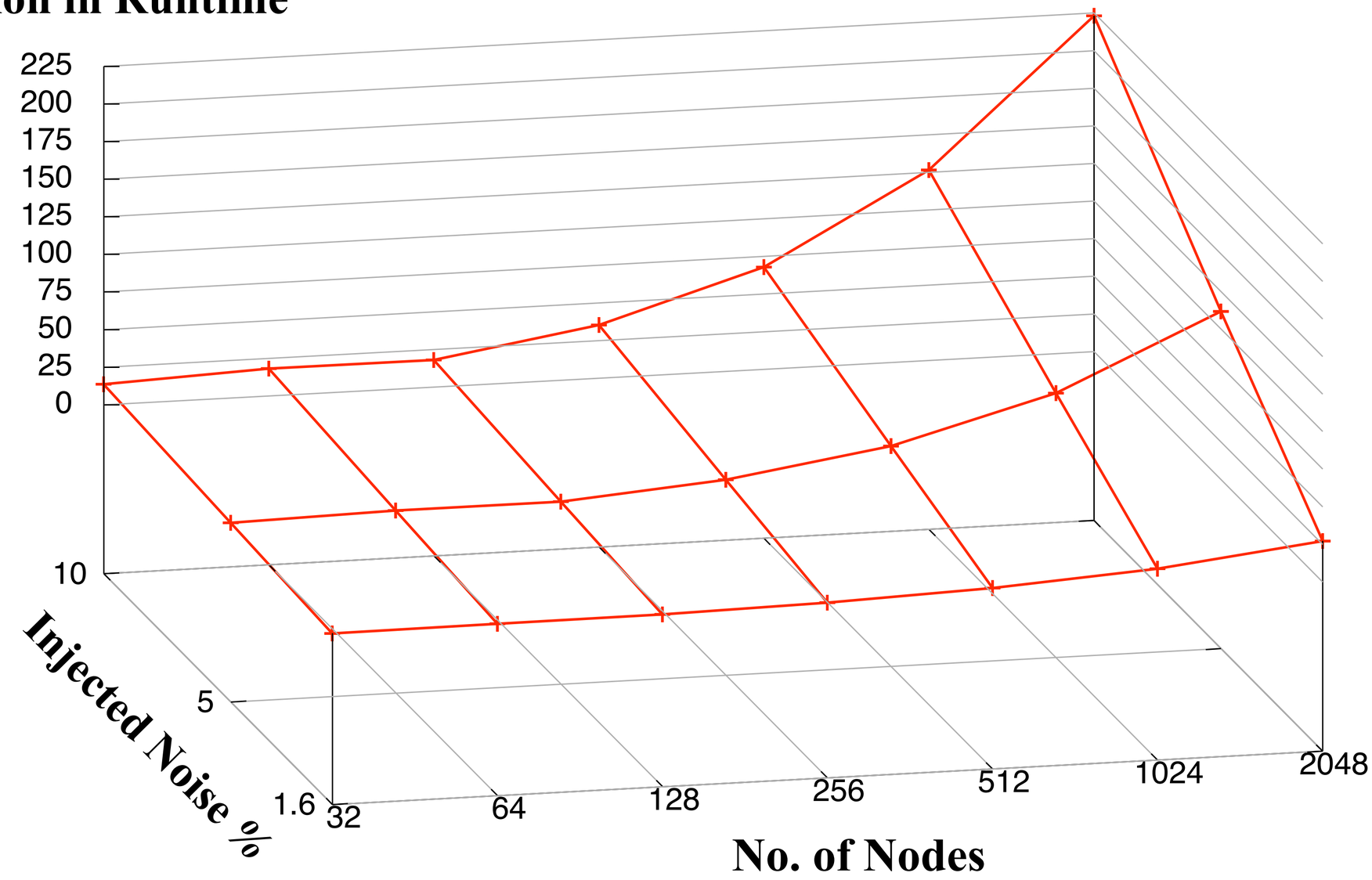
# *Accuracy of Noise-Estimation Analysis at Scale*

- How accurate is the "Accumulated Noise" value provided by analysis?
- Methodology
  - Take (relatively) noise-less platform (BG/L); Inject noise (Selfish Suite/ANL)
  - Run parallel application without noise & then with injected noise
  - Perform trace analysis to provide the *accumulated noise* estimate
  - How far is calculated *accumulated noise* value from actual delay?
- Simple BSP Benchmark (Bulk Synchronous Processing)
  - Repeated phases of computation & collective communication
  - Inputs: Scaling type, No. of Phases, Type of Collective, No. of Nodes
  - Used: Strong scaling, 10000 phases, Barrier, 32 to 2048 Nodes

# The Effect of Noise on the Benchmark

Noise Injection
Frequency: 1000 HZ
Length : 16, 50, 100 *u*sec

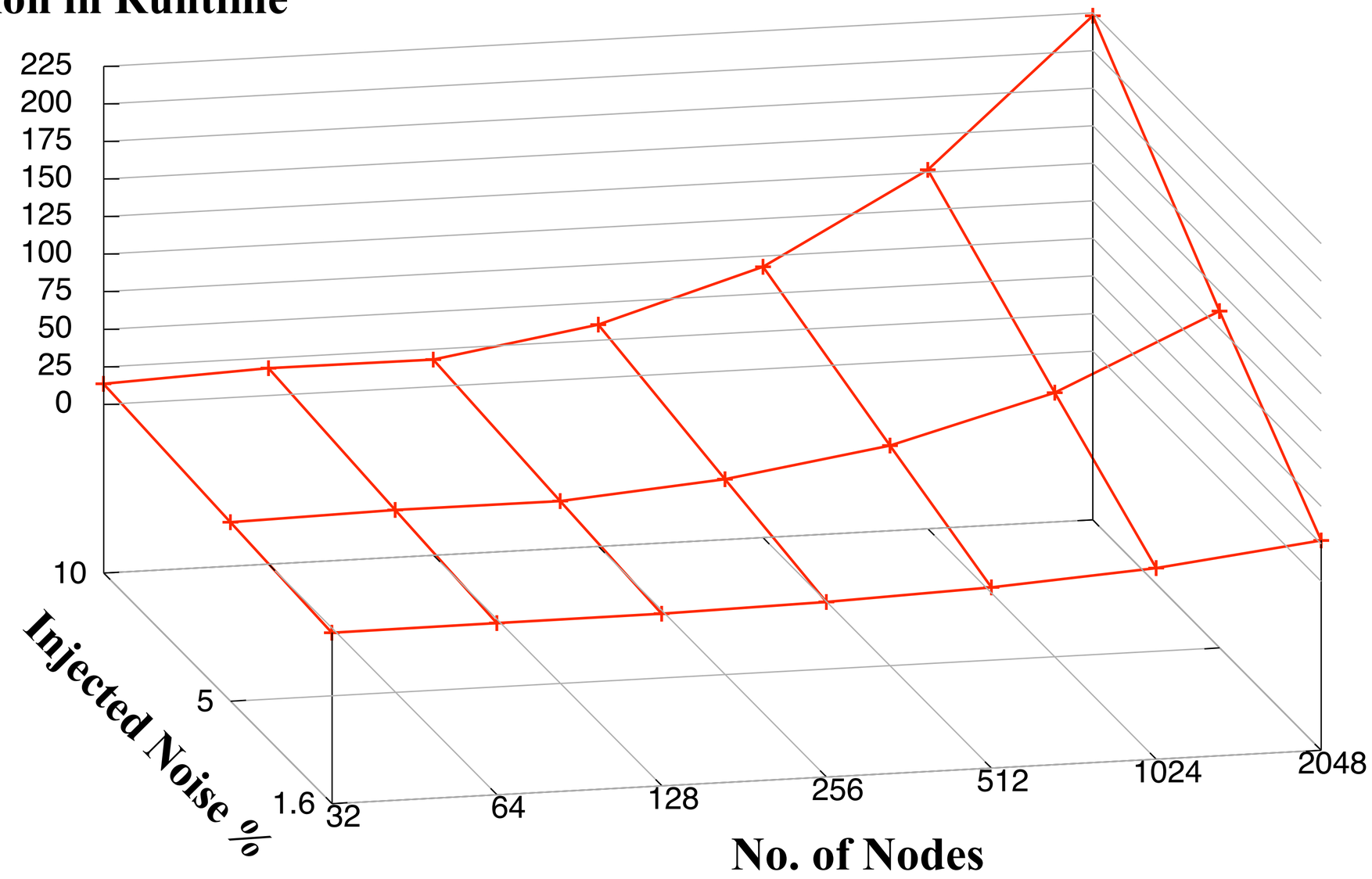
%Noise : 1.6%, 5%, 10%



# *The Effect of Noise on the Benchmark*

<u>Noise Injection</u>
Frequency: 1000 HZ
Length : 16, 50, 100 *u*sec

%Noise : 1.6%, 5%, 10%

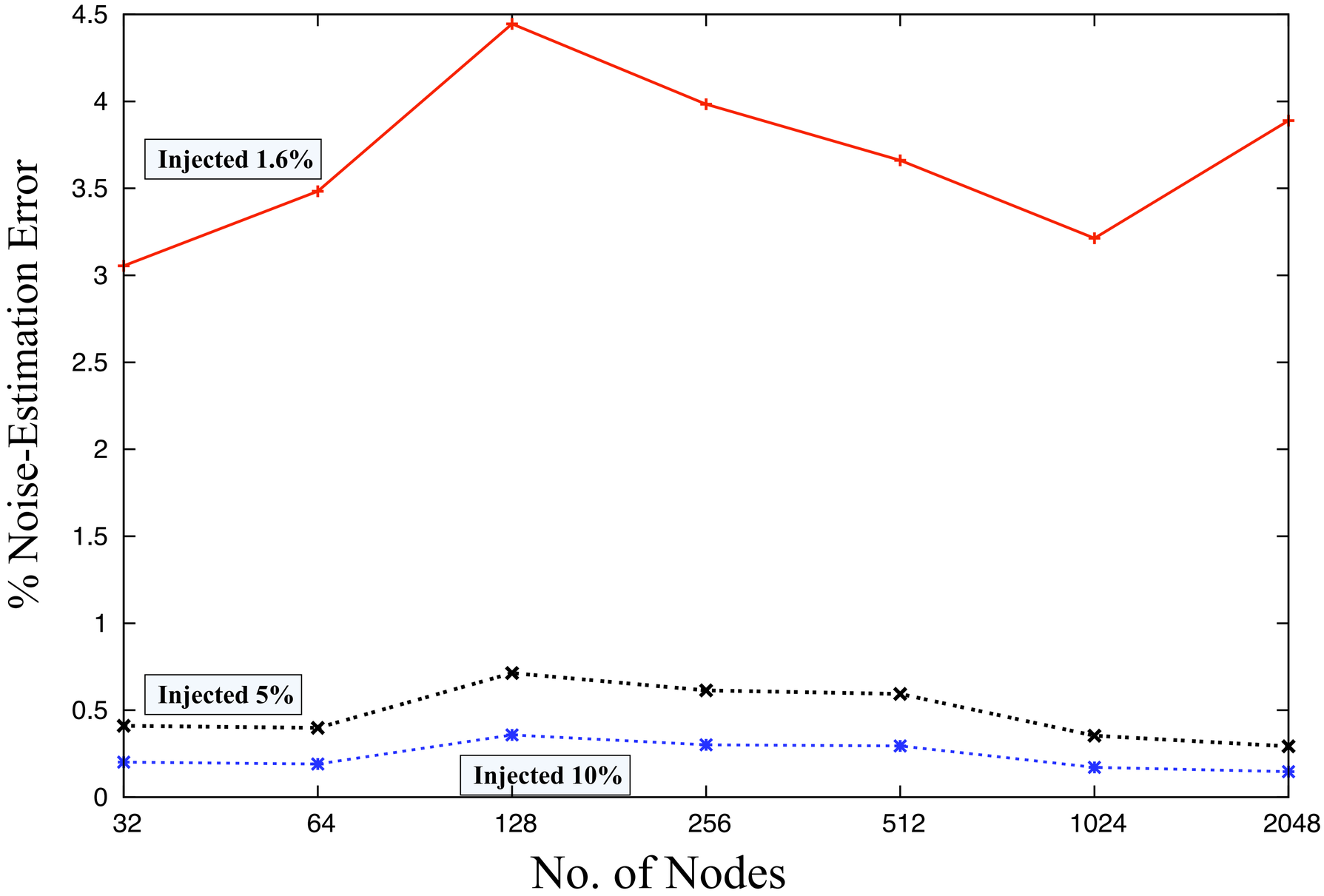



Perform Noise-Effect estimation for each trial.

# %Noise-Estimation Error

**%Noise-Estimation Error**

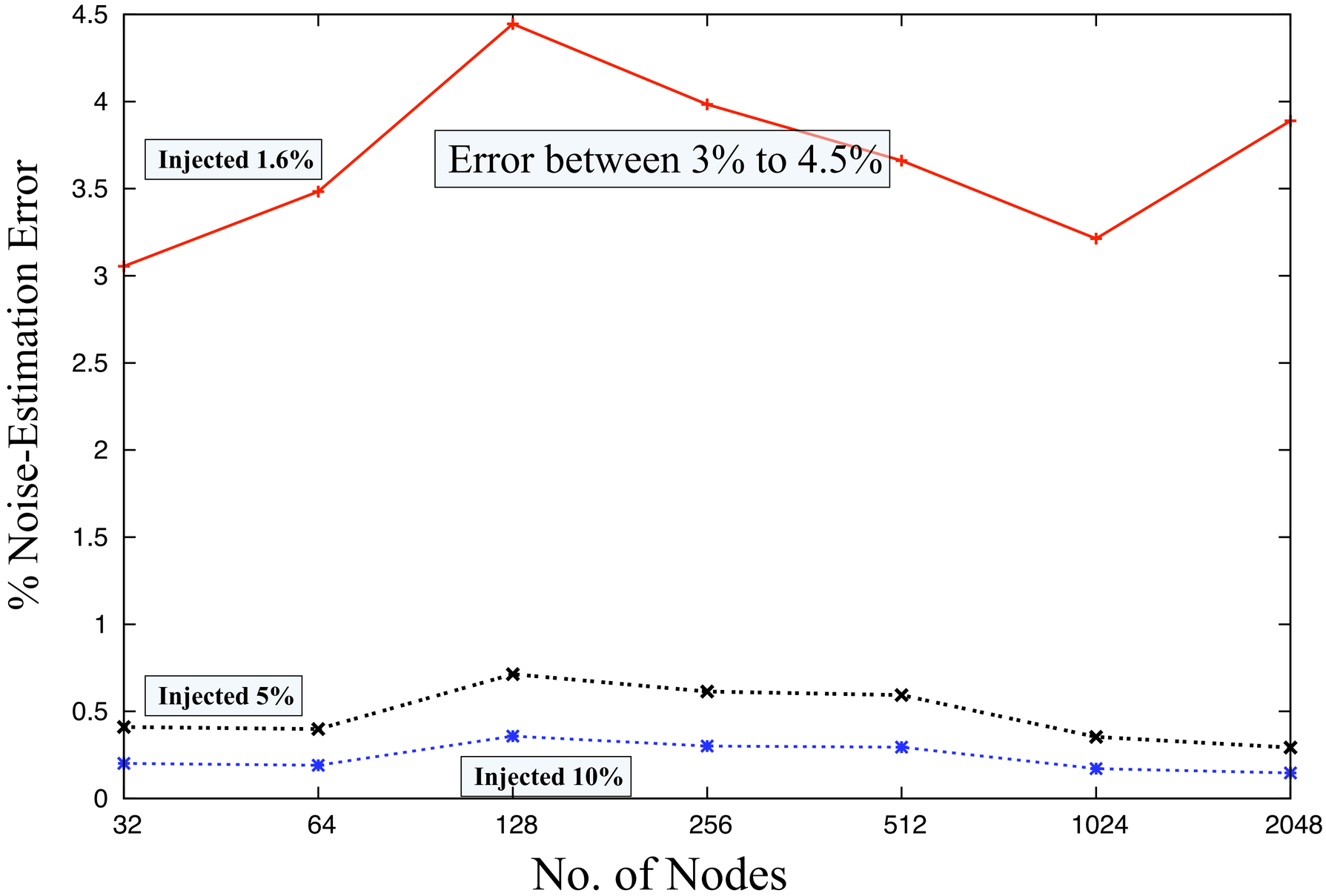
$$\frac{\text{Actual Noise - Acc. Noise}}{\text{Actual Noise}} * 100$$



# %Noise-Estimation Error

**%Noise-Estimation Error**

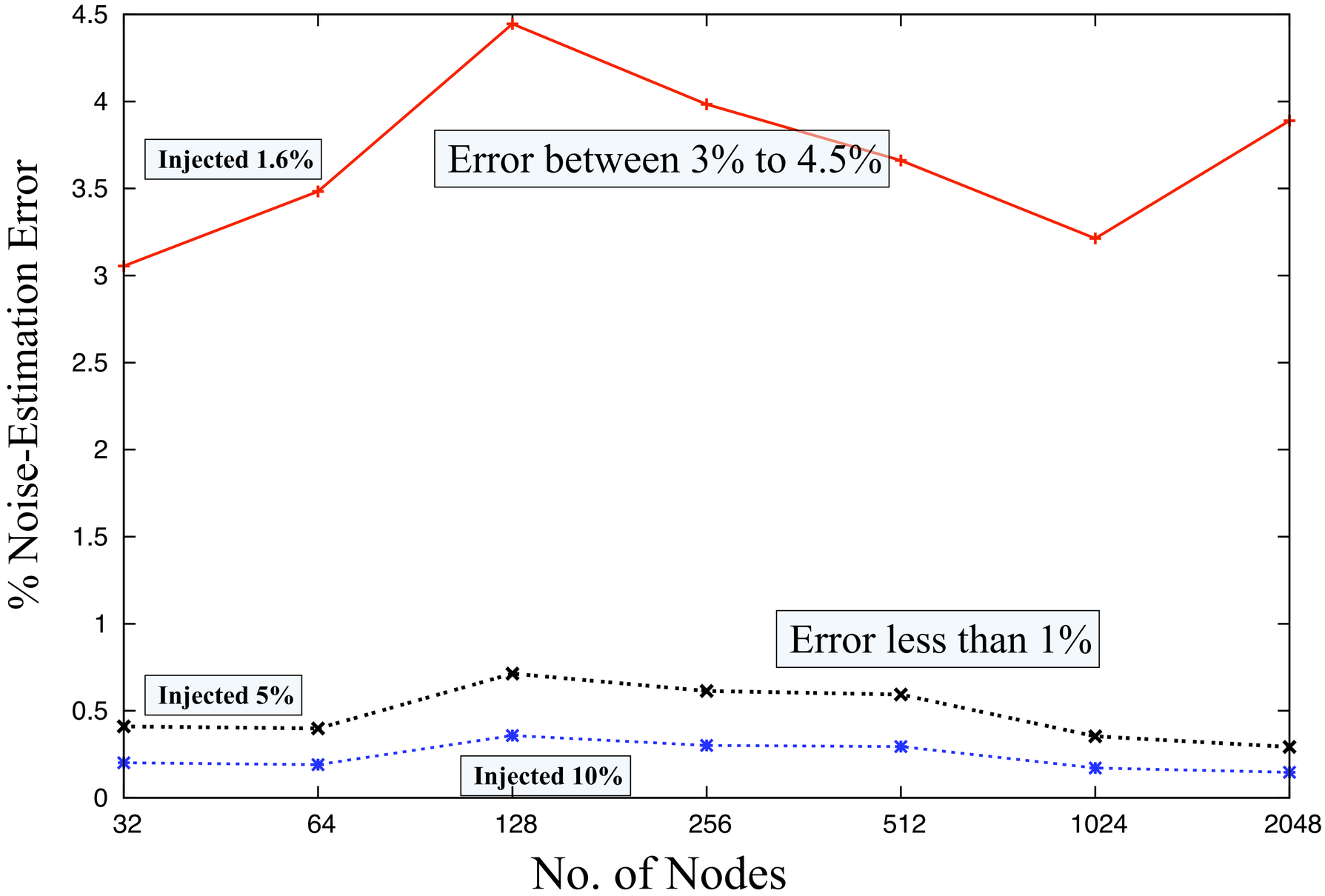
$$\frac{\text{Actual Noise - Acc. Noise}}{\text{Actual Noise}} * 100$$



# %Noise-Estimation Error

**%Noise-Estimation Error**

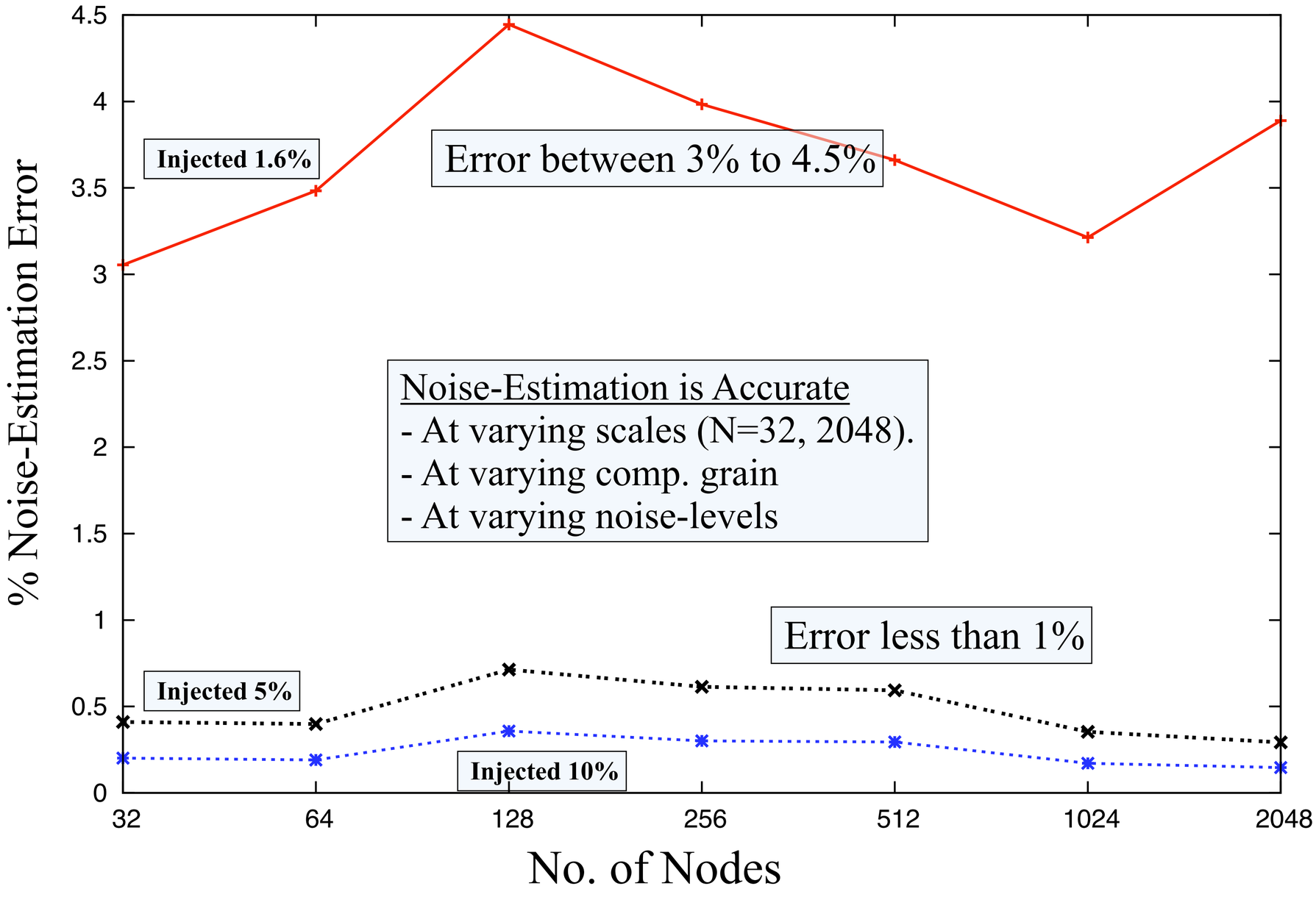
$$\frac{\text{Actual Noise - Acc. Noise}}{\text{Actual Noise}} * 100$$



# %Noise-Estimation Error

**%Noise-Estimation Error**

$$\frac{\text{Actual Noise - Acc. Noise}}{\text{Actual Noise}} * 100$$



# *How fast is access to KTAU OS Metrics?*

- Compare access costs (in cycles) to
  - PAPI counters
  - *no-op"* /proc call

| # Metrics | OS Metric Access | /proc Access | PAPI Access |
|---|---|---|---|
| 1 | **192** | **1150** | **248** |
| 2 | **272** | **-** | **304** |
| 3 | **288** | **-** | **-** |
| 4 | **400** | **-** | **-** |

- Metric Access order of magnitude faster than /proc access

- Metric Access comparable to PAPI h/w counter access

# *How fast is access to KTAU OS Metrics?*

- Compare access costs (in cycles) to
  - PAPI counters
  - *no-op"* /proc call

| # Metrics | OS Metric Access | /proc Access | PAPI Access |
|---|---|---|---|
| 1 | **192** | **1150** | **248** |
| 2 | **272** | **-** | **304** |
| 3 | **288** | **-** | **-** |
| 4 | **400** | **-** | **-** |

- Metric Access order of magnitude faster than /proc access

- Metric Access comparable to PAPI h/w counter access

# *How fast is access to KTAU OS Metrics?*

- Compare access costs (in cycles) to
  - PAPI counters
  - *no-op"* /proc call

| # Metrics | OS Metric Access | /proc Access | PAPI Access |
|---|---|---|---|
| 1 | **192** | **1150** | **248** |
| 2 | **272** | **-** | **304** |
| 3 | **288** | **-** | **-** |
| 4 | **400** | **-** | **-** |

- Metric Access order of magnitude faster than /proc access

- Metric Access comparable to PAPI h/w counter access

# *What is the Measurement Perturbation?*

- Measure Overall Perturbation of NPB LU under multiple configurations
  - Configuration: **base**
    - No instrumentation in application or OS
  - Configuration: ***ktau-tau-metrics***
    - TAU MPI tracing
    - Tracking 4 OS metrics for each MPI event

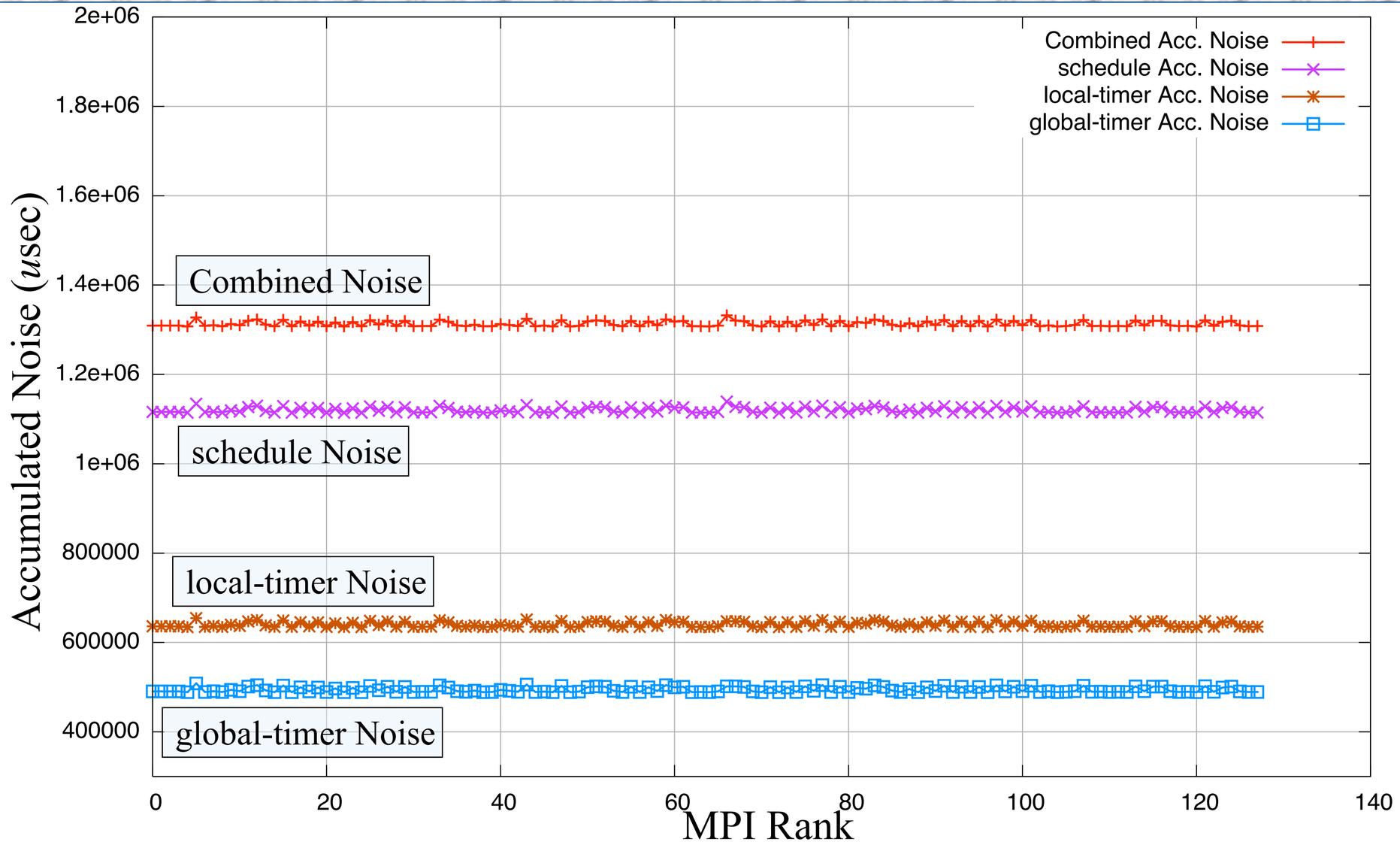
| **NPB LU Class C on 16 Nodes** | | |
|---|---|---|
| Configuration | **base** | ***ktau-tau-metrics*** |
| Minimum Exec. Time | **475.04** | **479.66** |
| % Min. Slowdown | **--** | **0.97** |

- **Performing OS Metric access + TAU MPI Tracing < 1% Perturbation**

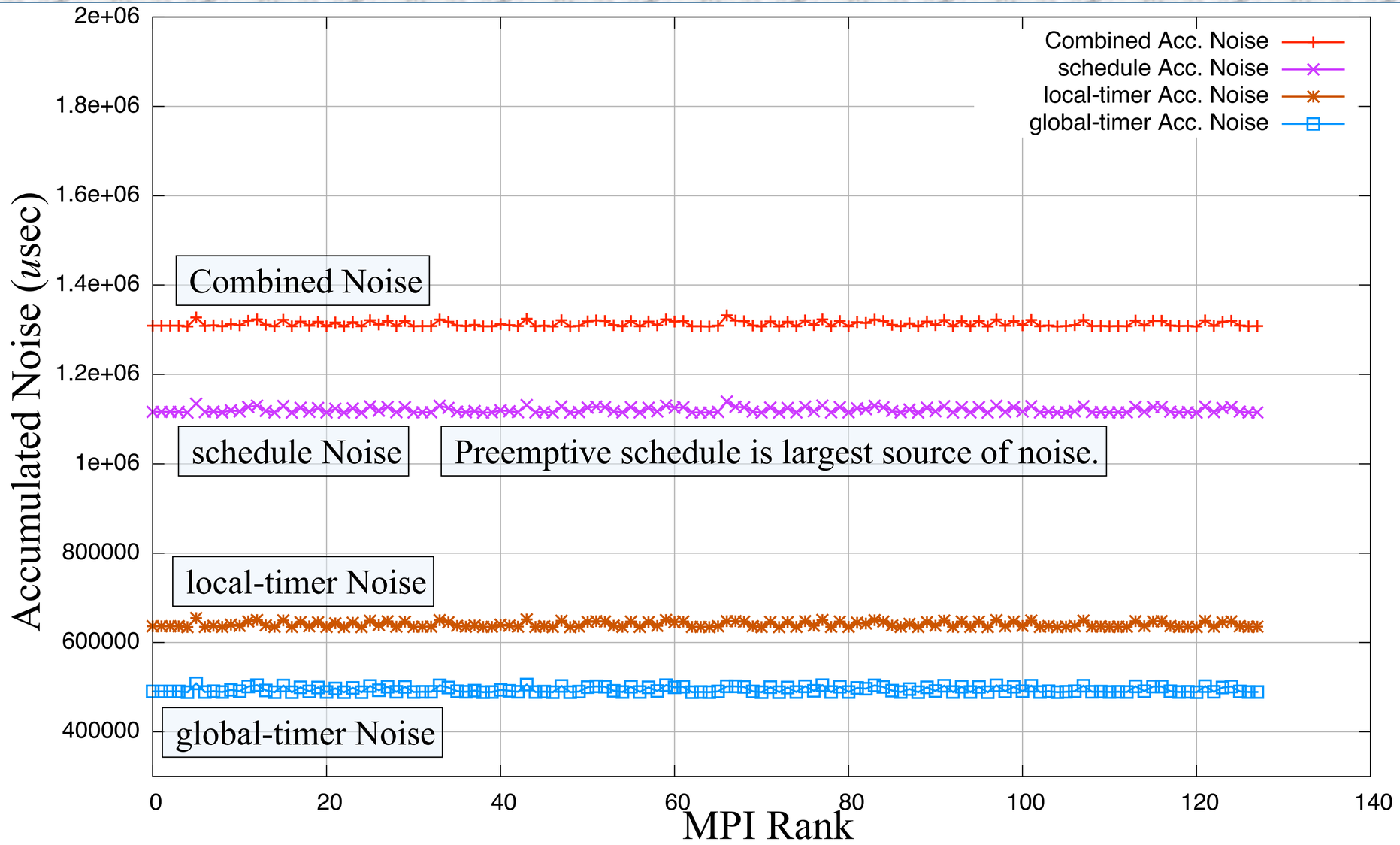
# *Conclusion*

- General measurement technique that estimates the delay caused by direct OS noise effects on a parallel message passing application
  - Integrated OS / Application performance measurement
  - Trace timeline approximation
- Demonstrate its use on a Linux cluster to measure effects of *real, existing noise*
- Evaluate accuracy of analysis at scale (2048 nodes)
  - 0.5% to 4.5% estimation error over varying noise levels and computational grain
- Questions in the HPC community
  - OS/R suites for large-scale platforms - Light-weight or Full-featured?
  - Can applications be changed to be less noise-sensitive?
  - DOE FAST-OS project created to investigate OS issues, including noise
  - Integrated (OS/Application) measurement and noise analysis techniques can aid in answering some of the questions

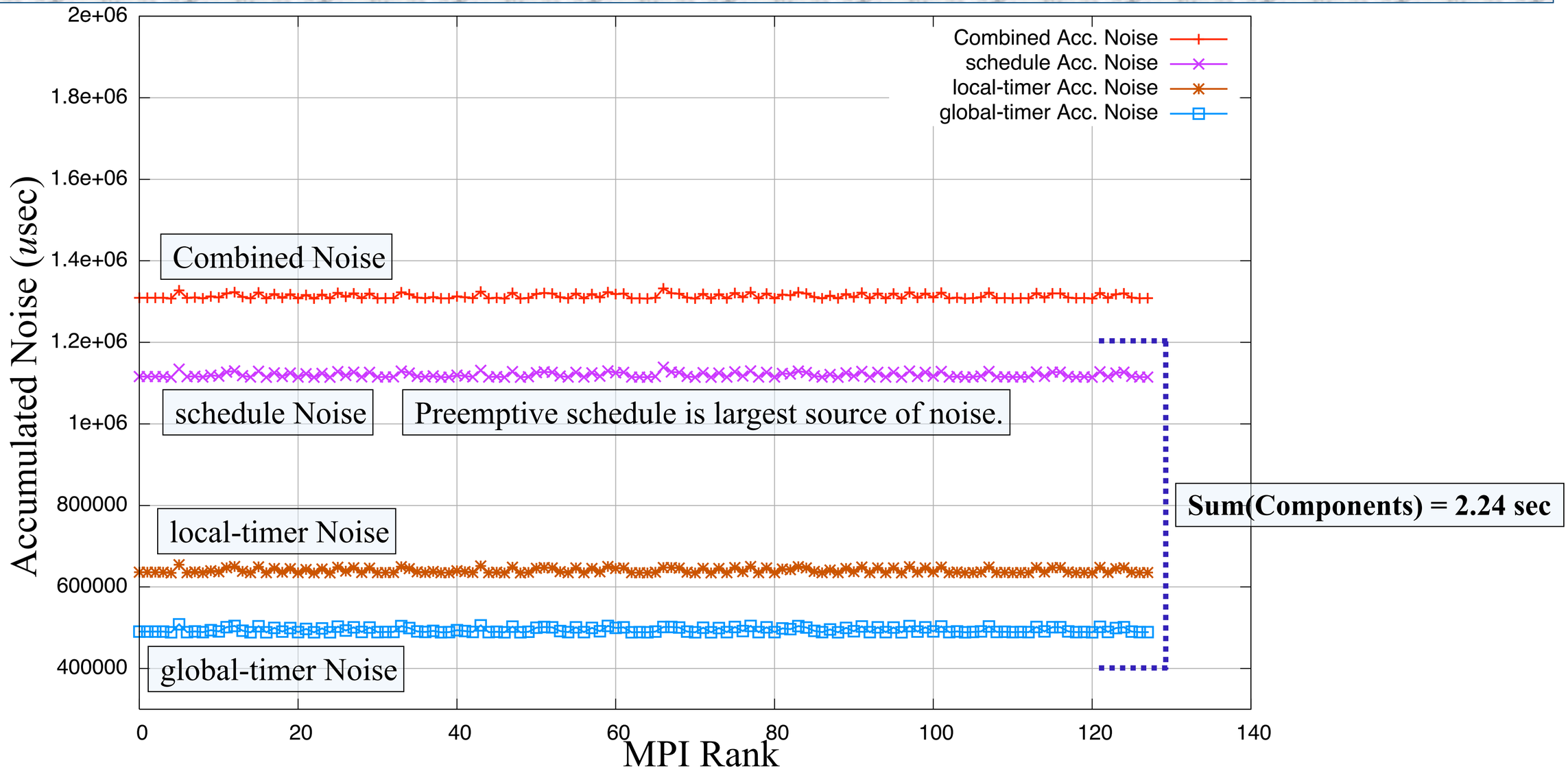
# Combining Noise Sources -- Accumulated Noise -- P=128



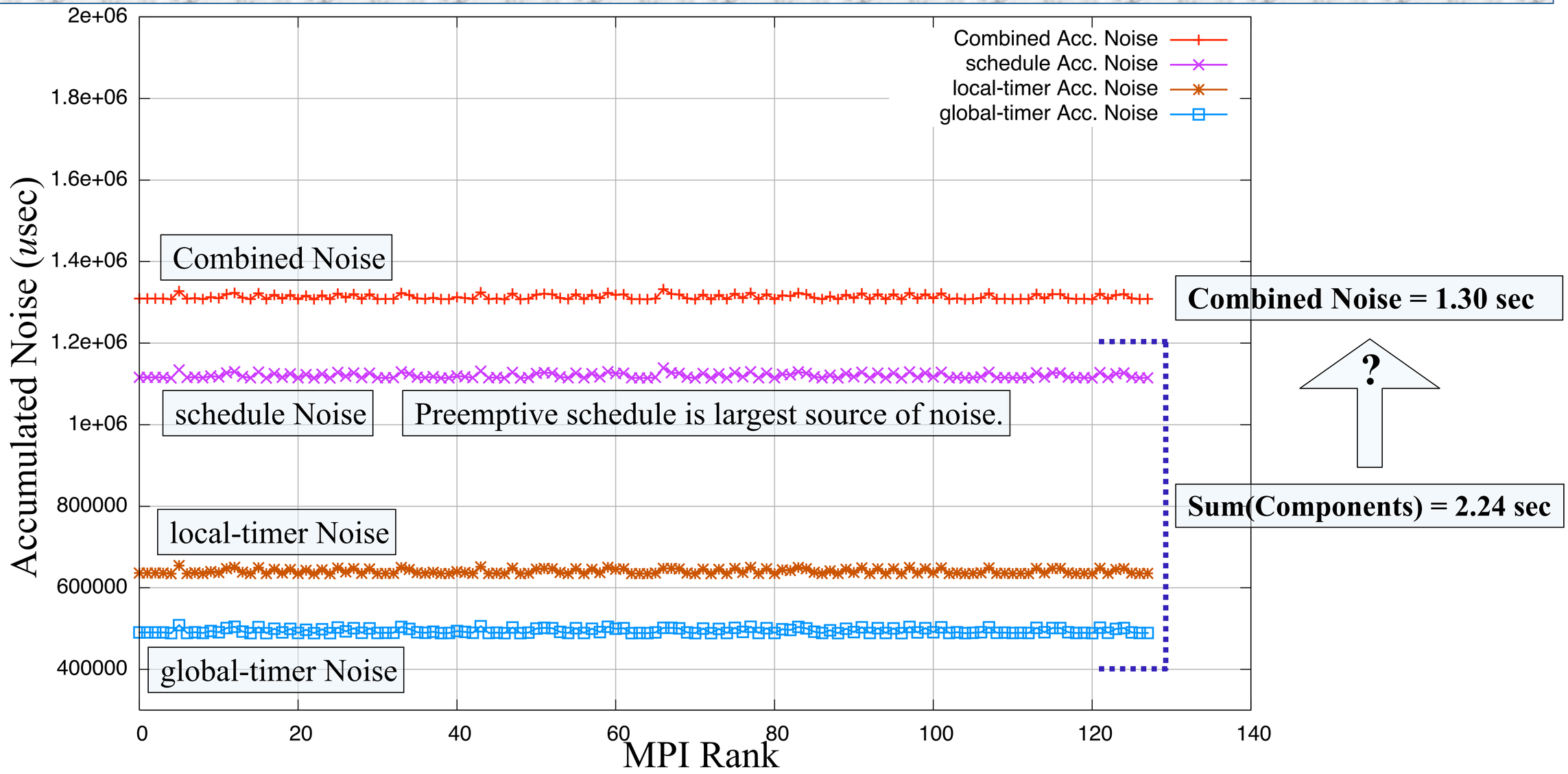
# Combining Noise Sources -- Accumulated Noise -- P=128



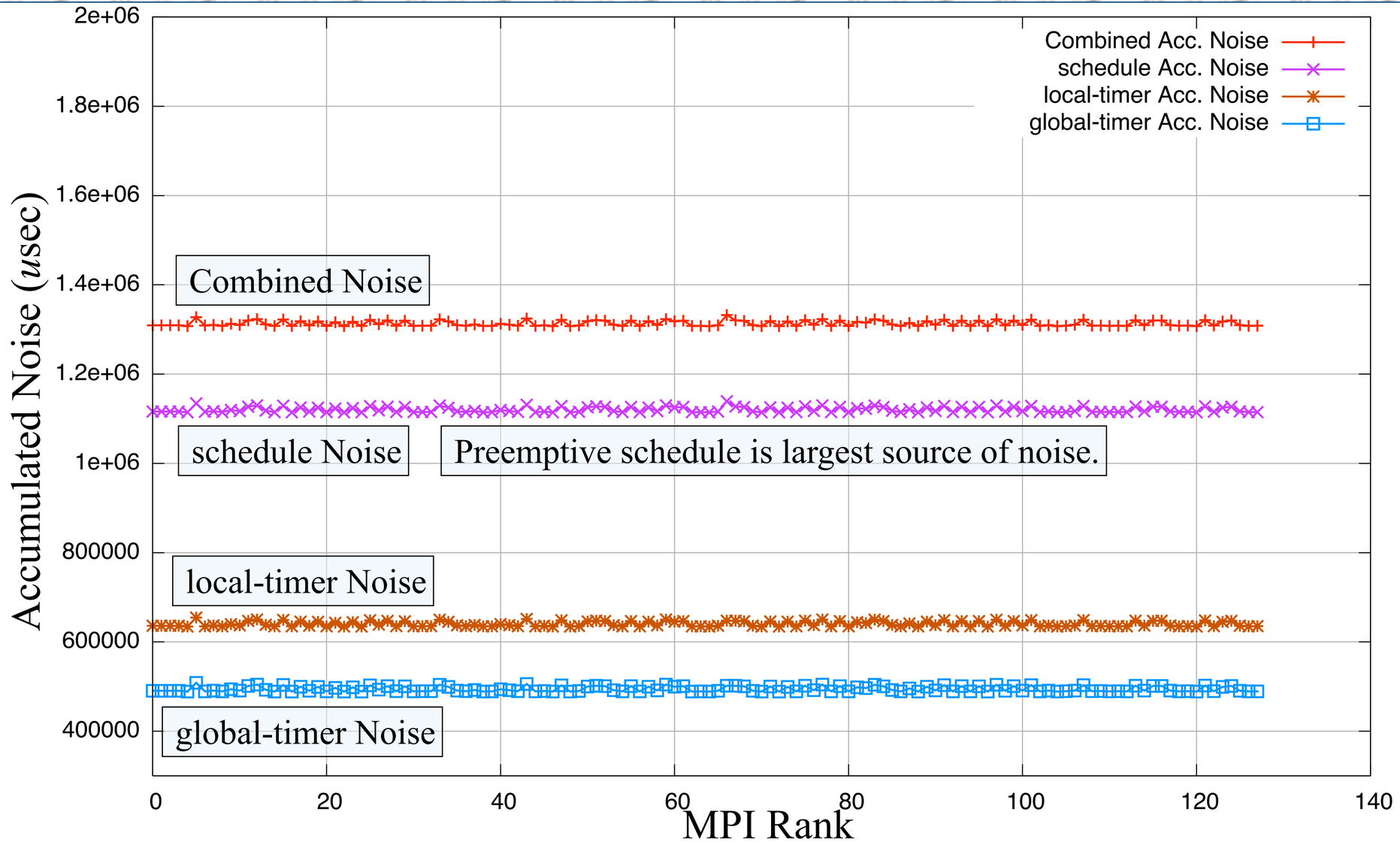
# Combining Noise Sources -- Accumulated Noise -- P=128



# *Combining Noise Sources -- Accumulated Noise -- P=128*



# Combining Noise Sources -- Accumulated Noise -- P=128




***Why?***

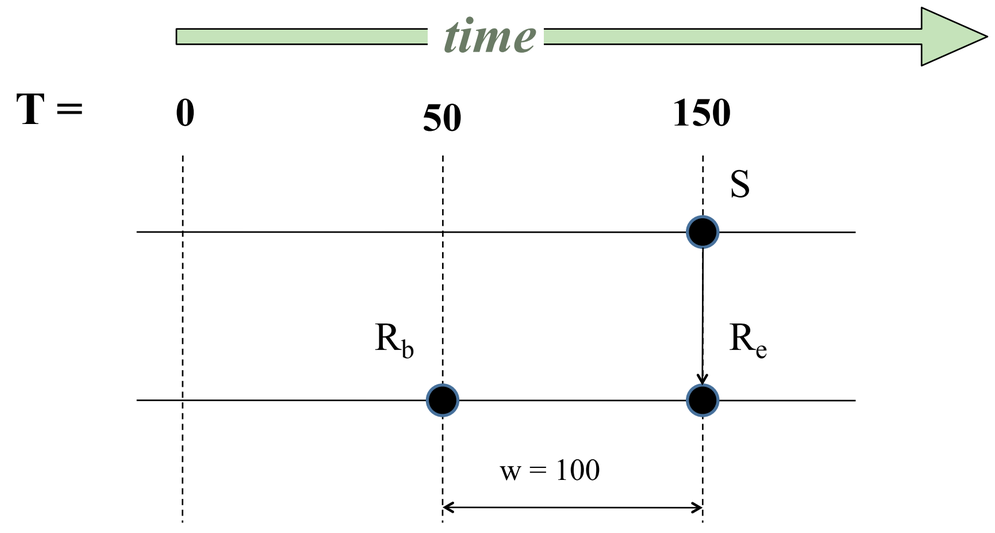
# Global Noise Estimation - Noise Combination

Process 1 Sends; Process 2 Receives

## Measured Timeline

**Process 1**
***Local* Noise $N_1$ = 80s**

**Process 2**
***Local* Noise $N_1$ = 0s**




## Approximated Timeline

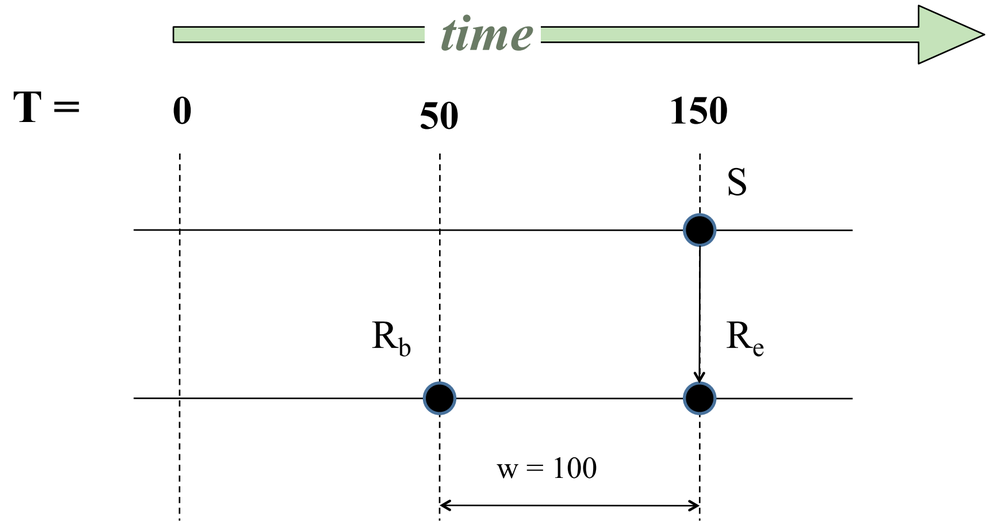
# Global Noise Estimation - Noise Combination

Process 1 Sends; Process 2 Receives

## Measured Timeline

**Process 1**
***Local* Noise $N_1$ = 80s**

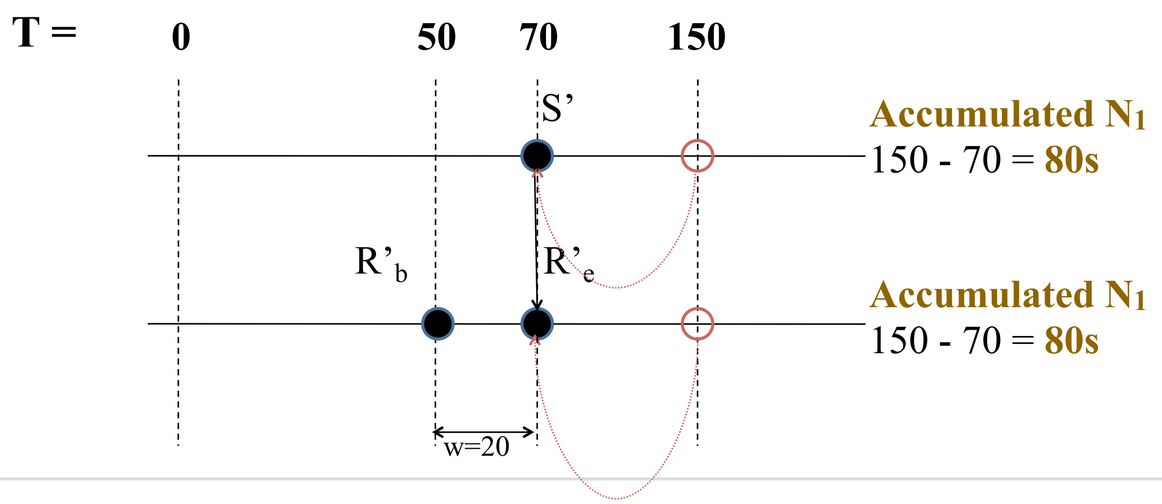
**Process 2**
***Local* Noise $N_1$ = 0s**




## Approximated Timeline

**Process 1**
***Local* Noise $N_1$ = 80s**

**Process 2**
**Local Noise $N_1$ = 0s**



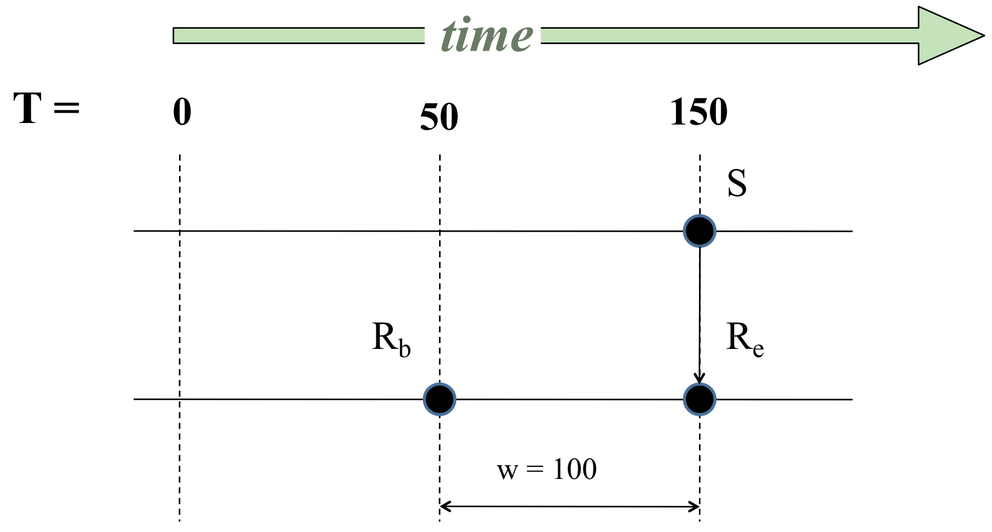
# Global Noise Estimation - Noise Combination

Process 1 Sends; Process 2 Receives

## Measured Timeline

**Process 1**
***Local* Noise $N_2$ = 70s**

**Process 2**
**Local Noise $N_2$ = 0s**




## Approximated Timeline

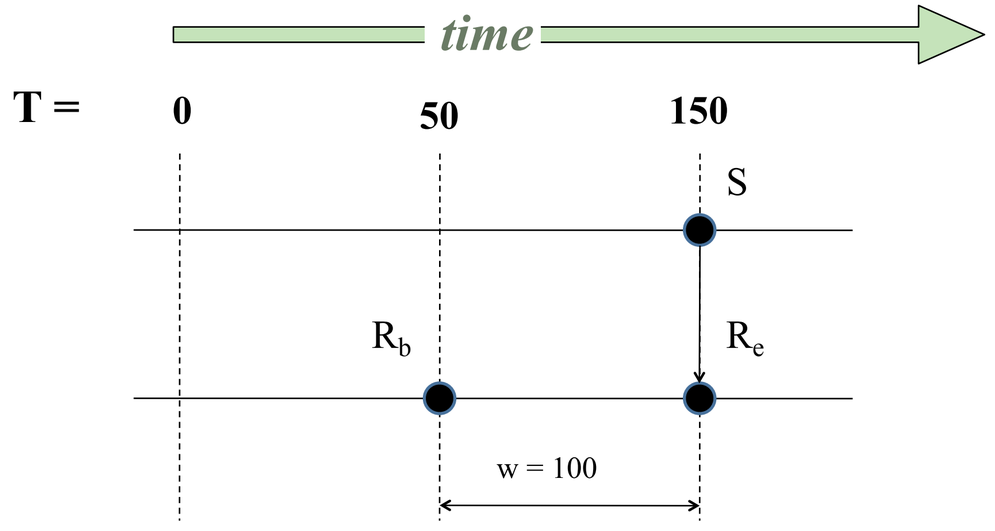
# Global Noise Estimation - Noise Combination

Process 1 Sends; Process 2 Receives

## Measured Timeline

**Process 1**
***Local* Noise $N_2$ = 70s**

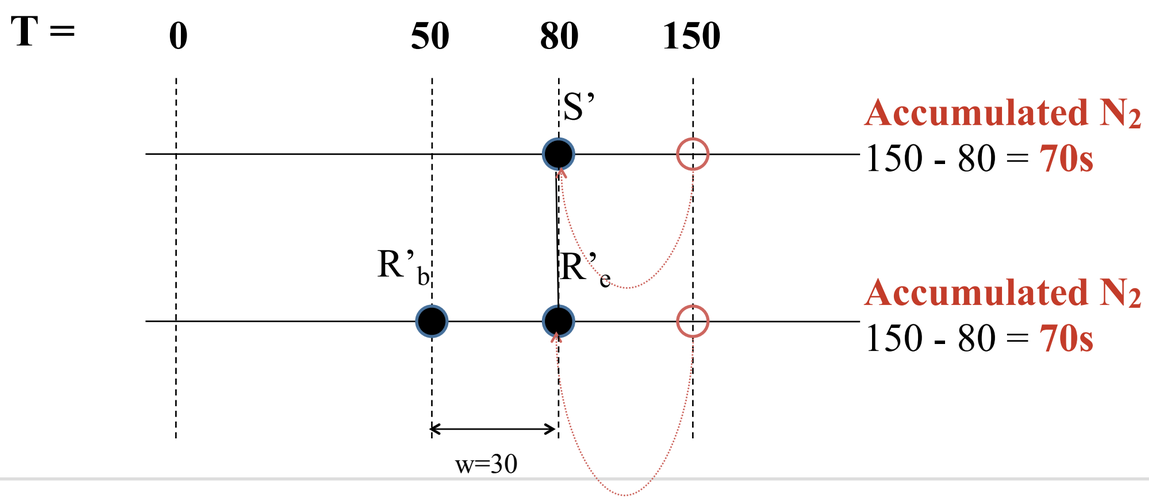
**Process 2**
**Local Noise $N_2$ = 0s**




## Approximated Timeline

**Process 1**
***Local* Noise $N_2$ = 70s**

**Process 2**
**Local Noise $N_2$ = 0s**



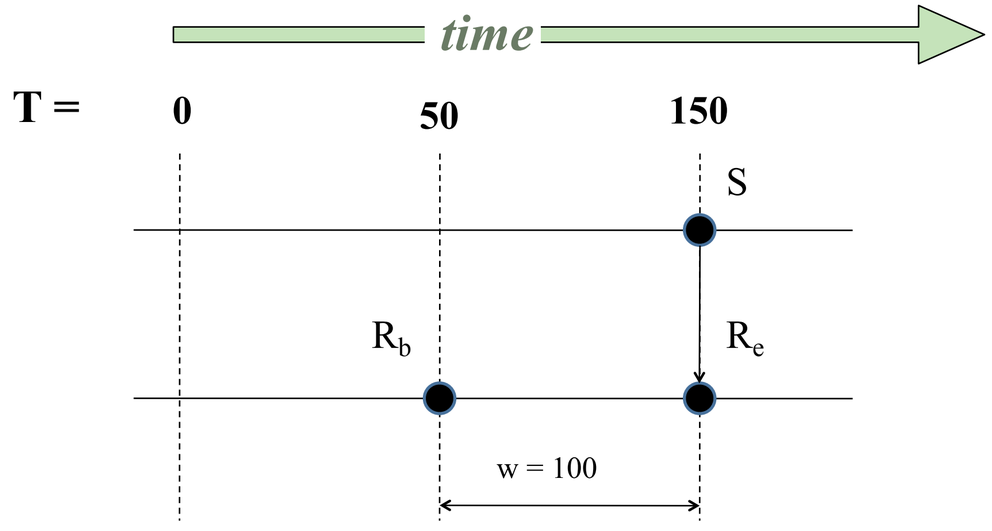
# Global Noise Estimation - Noise Combination

Process 1 Sends; Process 2 Receives

## Measured Timeline

**Process 1**
***Local* Noise $N_1+N_2$ =150s**

**Process 2**
***Local* Noise $N_1+N_2$ =0s**




## Approximated Timeline

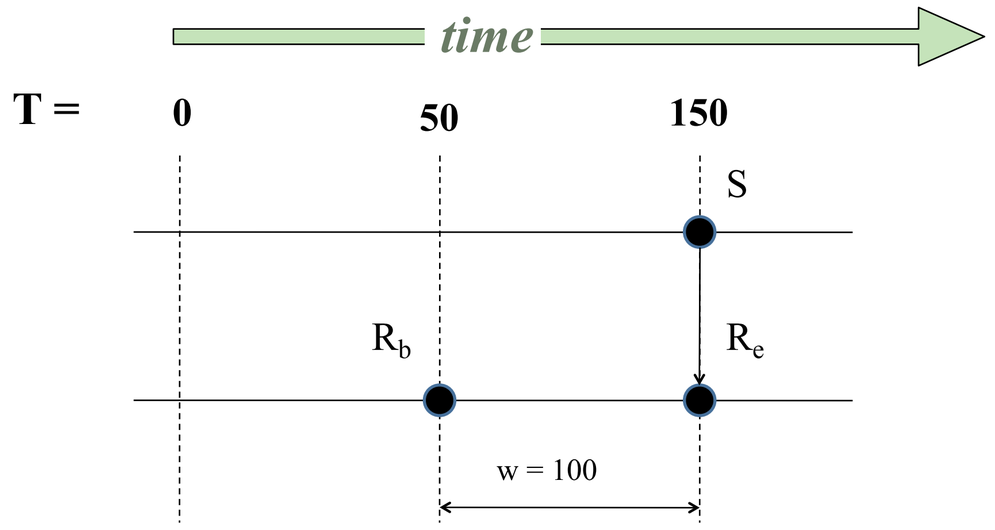
# Global Noise Estimation - Noise Combination

Process 1 Sends; Process 2 Receives

## Measured Timeline

**Process 1**
***Local* Noise $N_1+N_2$ =150s**

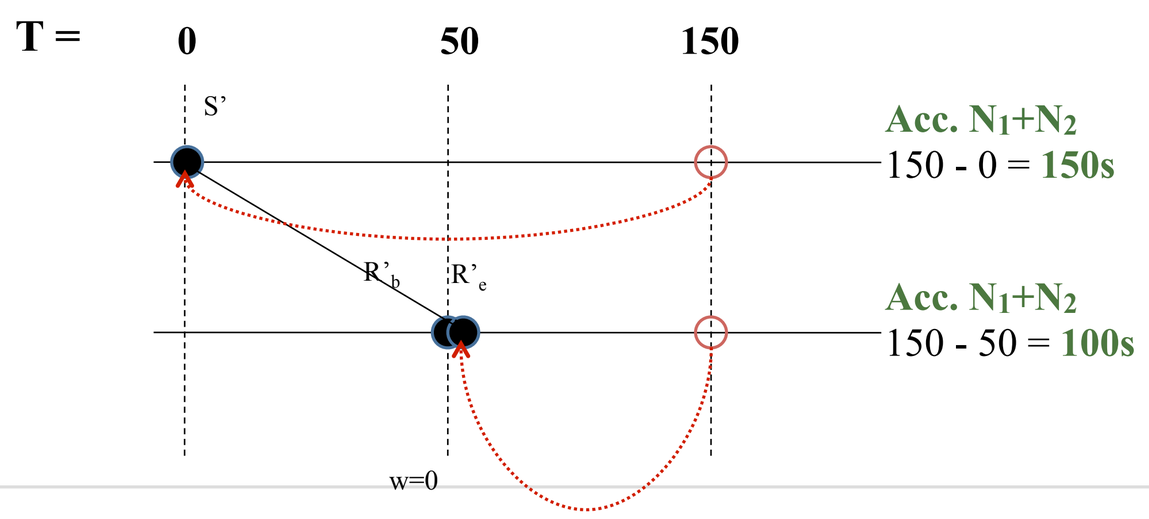
**Process 2**
***Local* Noise $N_1+N_2$ =0s**



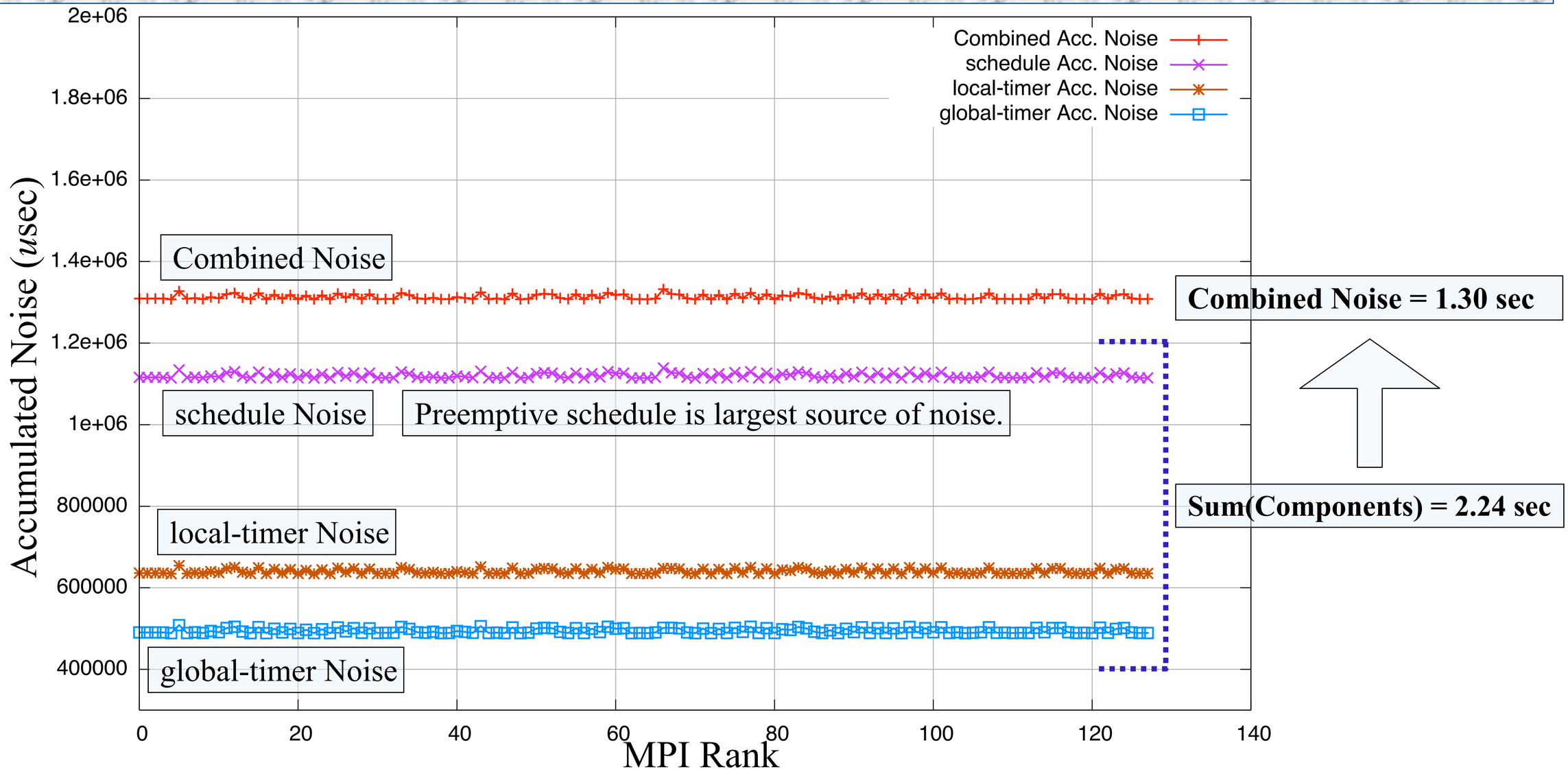

## Approximated Timeline

**Process 1**
***Local* Noise $N_1+N_2$ =150s**

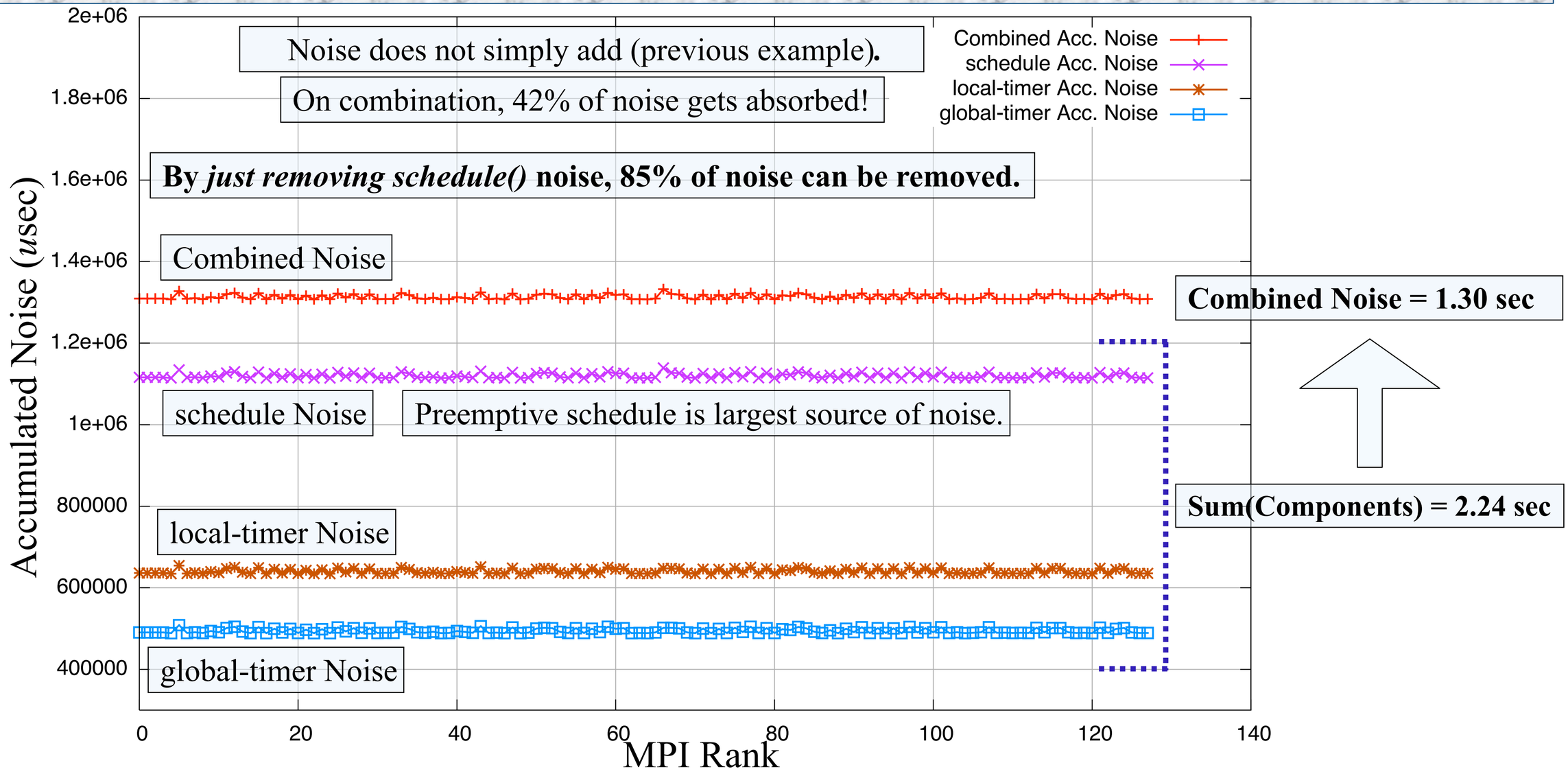
**Process 1**
***Local* Noise $N_1+N_2$ =0s**



# *The Noise Sources -- Accumulated Noise -- P=128*



# *The Noise Sources -- Accumulated Noise -- P=128*



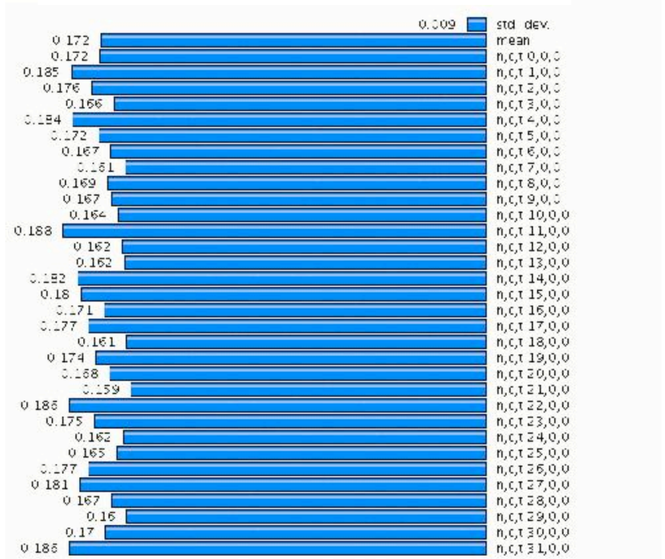
# *Related Work in Noise Measurement*

- Petrini et al. SC'03 - Microbenchmark, simulation, modeling to *close the loop*
  - Specific to application (SAGE). Accurate model difficult to produce.
- Gioiosa et al. ISSIPIT'04 - Microbenchmark & measurement
  - Identify noise sources using OProfile sampling. Quantifies only *local-noise*.
- Agarwal et al. HiPC'05 - Theoretical Modeling under different distributions
  - Assumptions include: Balanced Load, Stationary, Balanced Noise, Identical noise
- Beckman et al. CLUSTER'06 (Emulation), Sottile et al. IPDPS'06 (Simulation)
  - Injected artificial noise at runtime into micro-benchmarks to understand effects
  - Modify application traces by adding artificial noise.

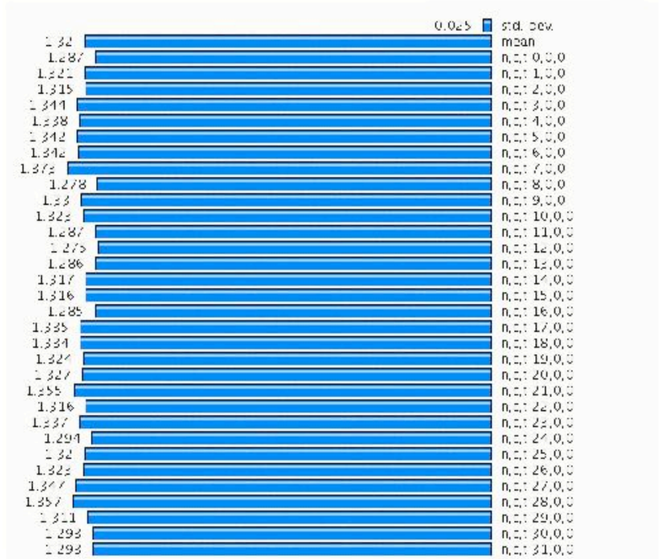
# *Profiling LU Application using KTAU OS Metrics*

Noise metrics per process

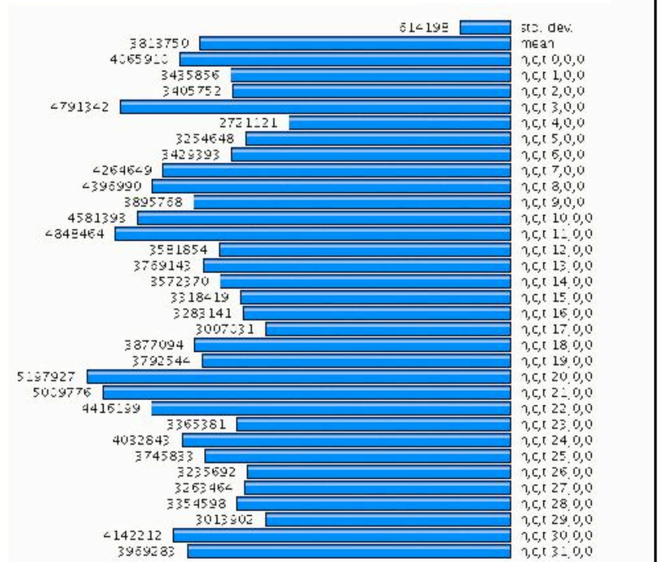
*global timer interrupt*




*local timer interrupt*



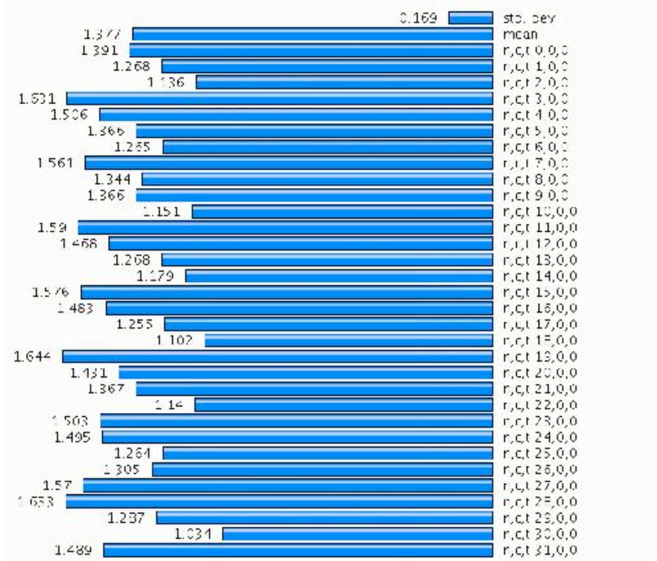

*schedule*



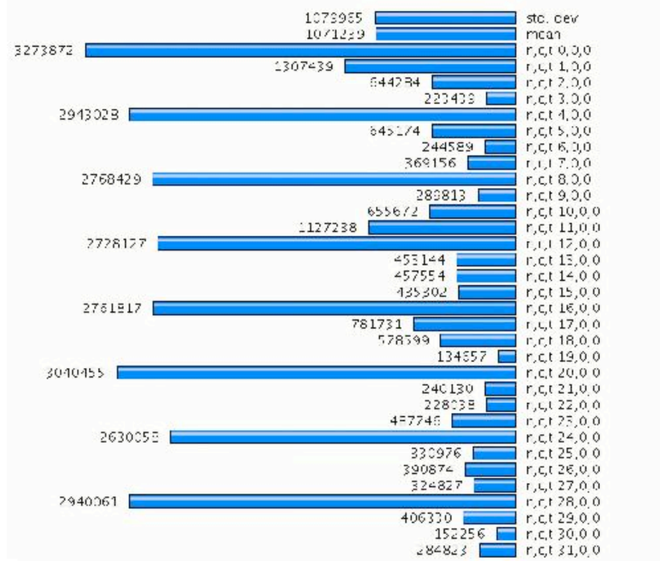

No irq balance and pinning

*global timer interrupt*


*local timer interrupt*




*schedule*

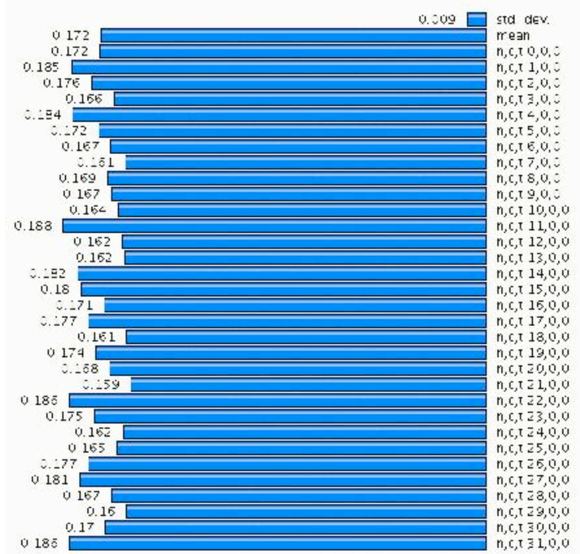


# *Profiling LU Application using KTAU OS Metrics*

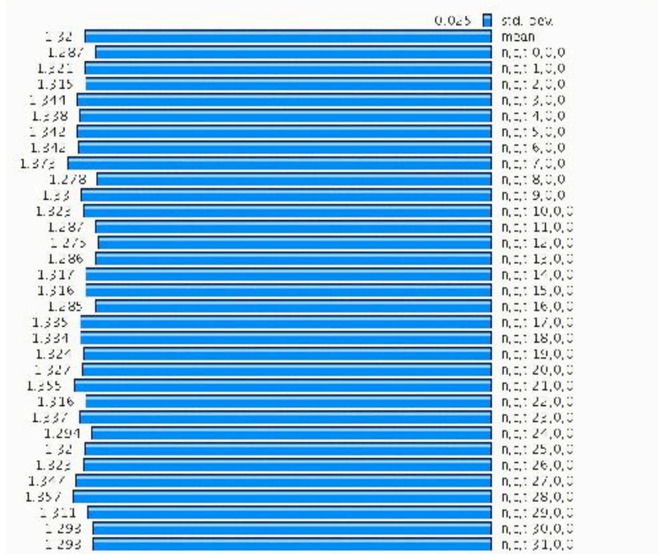
Noise metrics per process

No irq balance and pinning

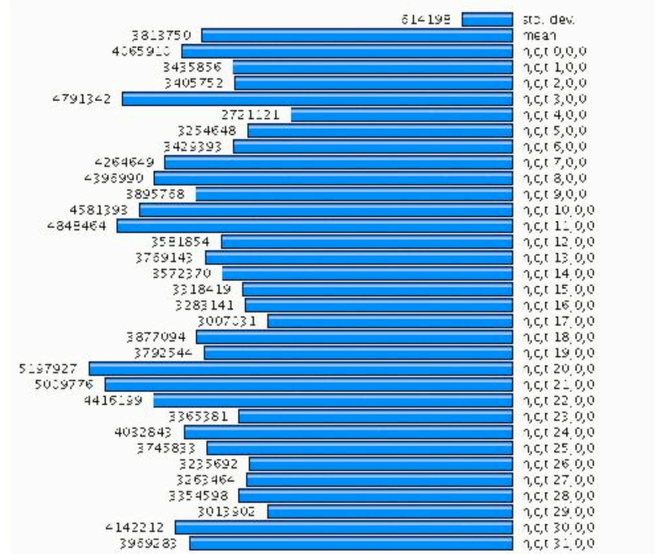
*global timer interrupt*



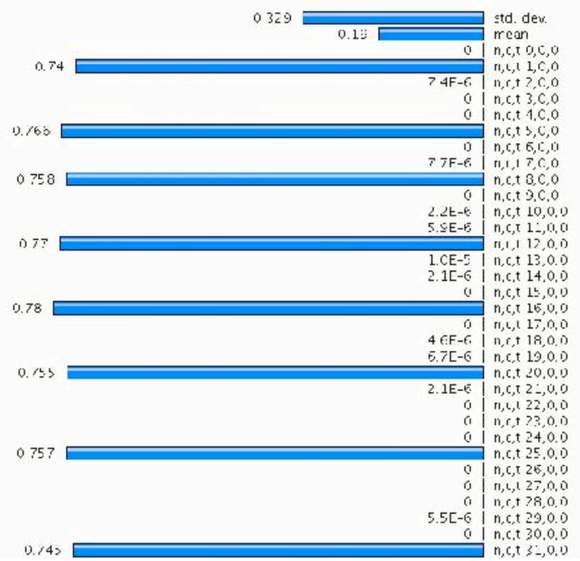

*local timer interrupt*



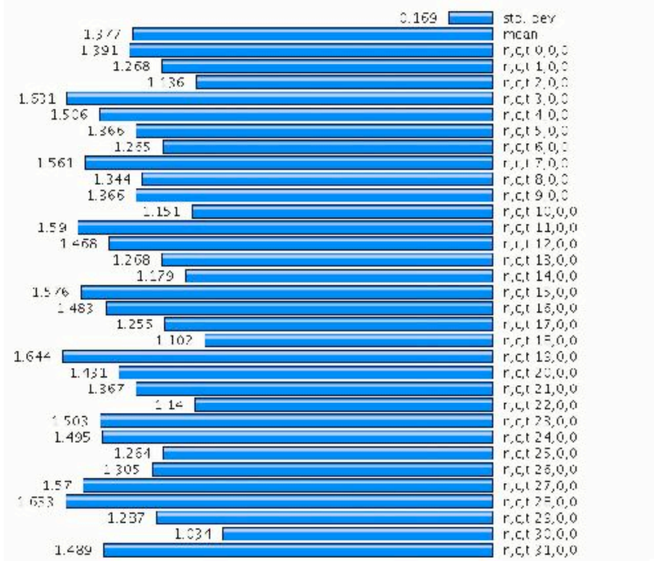

*schedule*



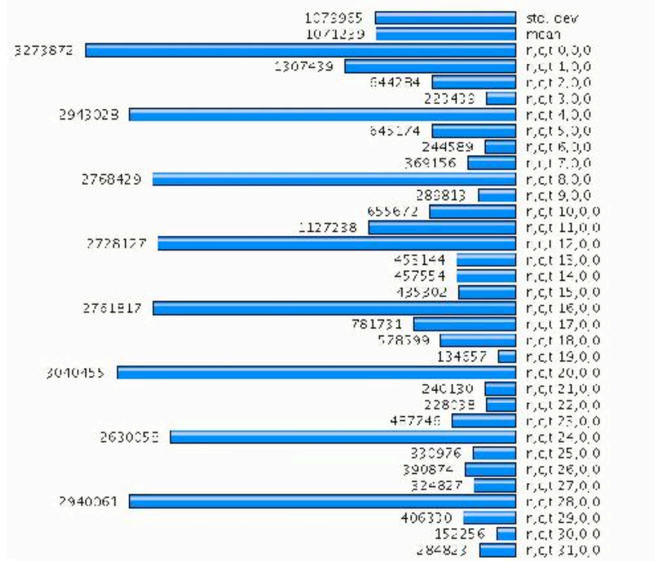

*global timer interrupt*




*local timer interrupt*



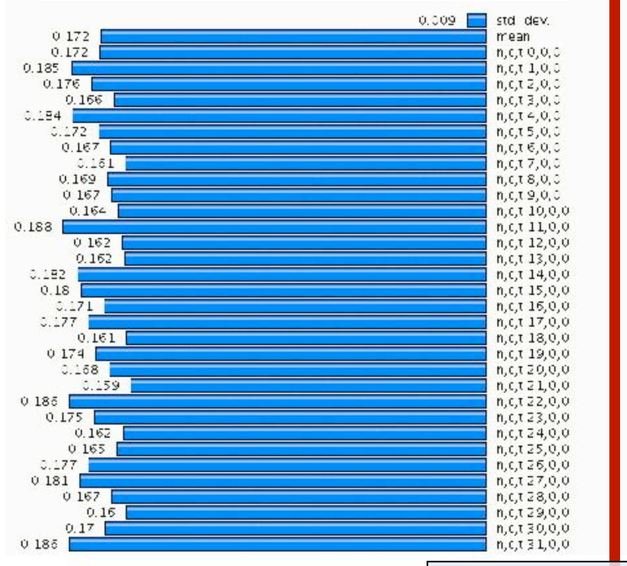

*schedule*



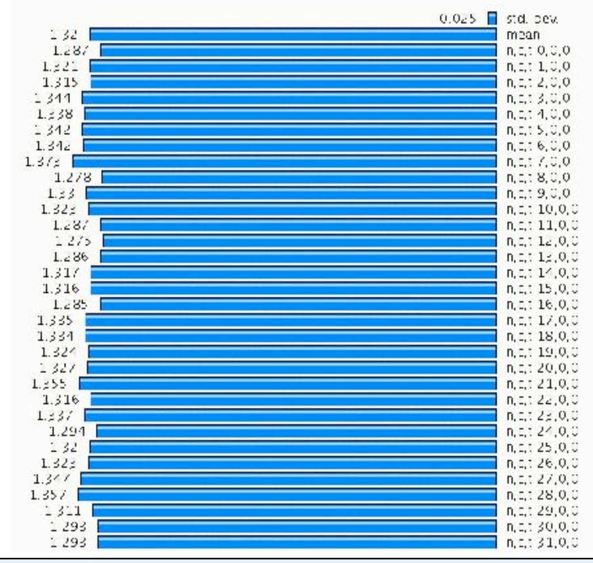
# *Profiling LU Application using KTAU OS Metrics*

Noise metrics per process

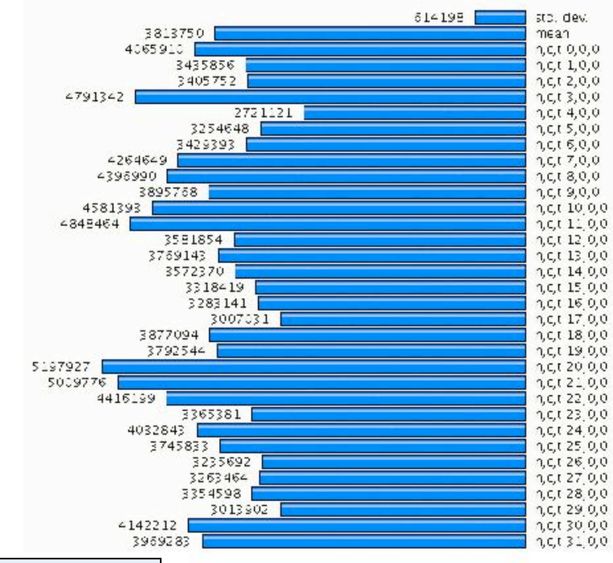
*global timer interrupt*




*local timer interrupt*



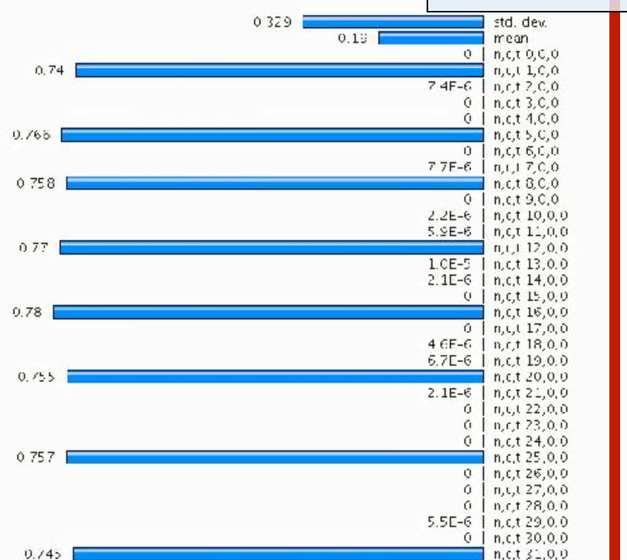

*schedule*



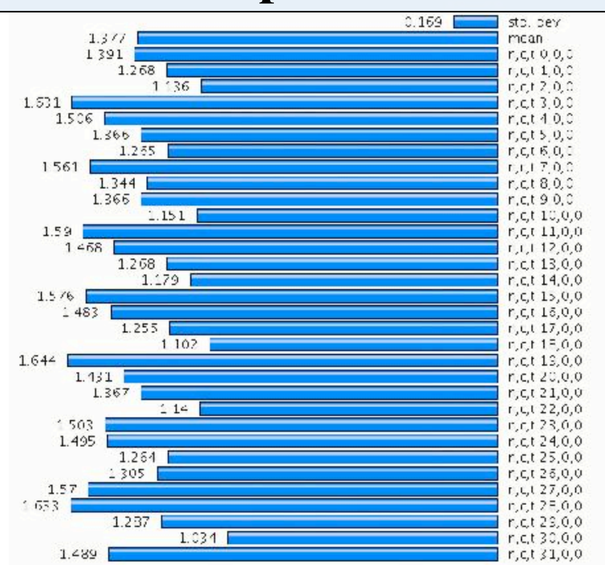

**_Local noise_ is measured and _attributed_ to respective MPI ranks.**

No irq balance and pinning

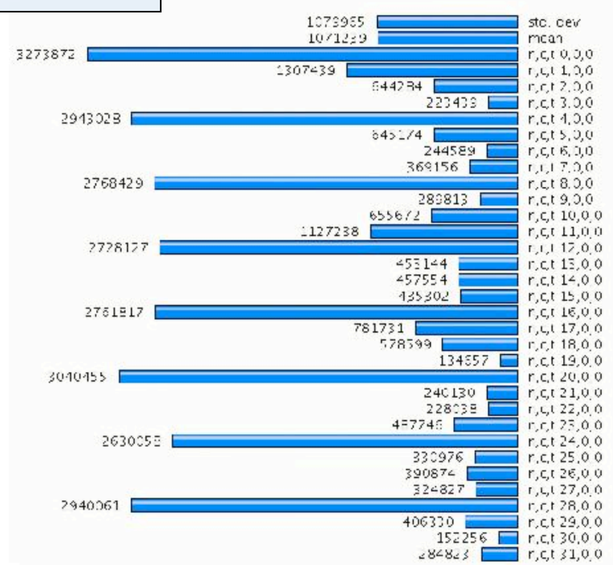
*global timer interrupt*




*local timer interrupt*

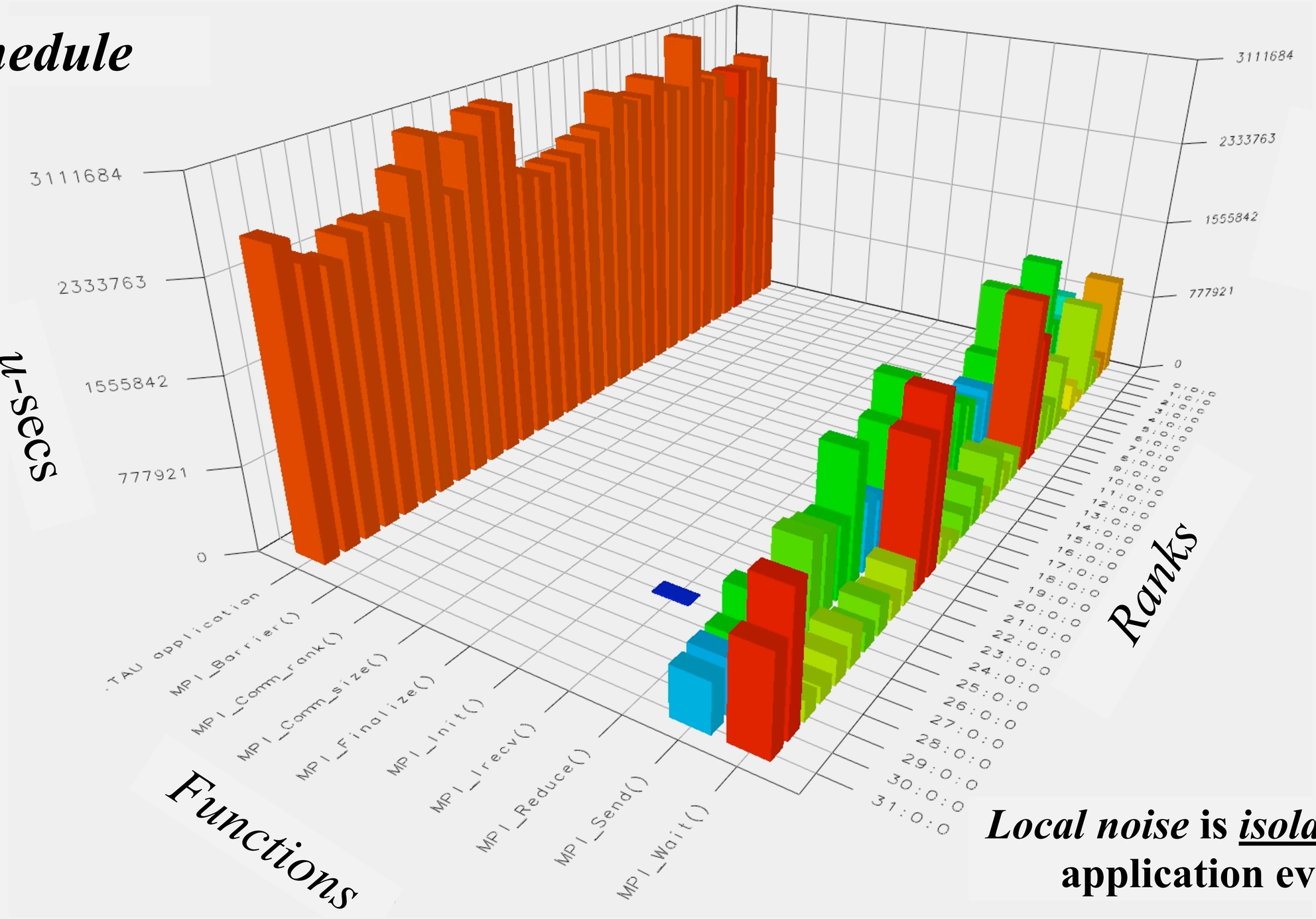



*schedule*



# *Profiling CG using KTAU OS Metrics*

## *schedule*




***Local noise* is *isolated* from application events.**

# *Acknowledgments*


- San Diego Supercomputing Center (SDSC) & Don Thorp
  - Access to Opteron cluster
- Argonne National Laboratory
  - Access to BG/L and other compute resources
- DOE FAST-OS Project
  - Funding as part of the joint ZeptoOS project between ANL and UO