Characterizing Technical Software Performance Within System of Systems Acquisitions: *A Step-Wise Methodology*

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

B. Meyer, J. Wessel
May 2010

Introduction

System of systems (SoS), either directed as a program, acknowledged as a set of programs, or emergent as in collaborative or virtual varieties*, ALL need a way to assess software performance (SWP):

- Assess causes of SWP issues
- Determine indicators and measures of SWP
- Plan SWP measurement in tests

Fundamental question: *Will software enable planned capabilities within end-to-end field environment?*

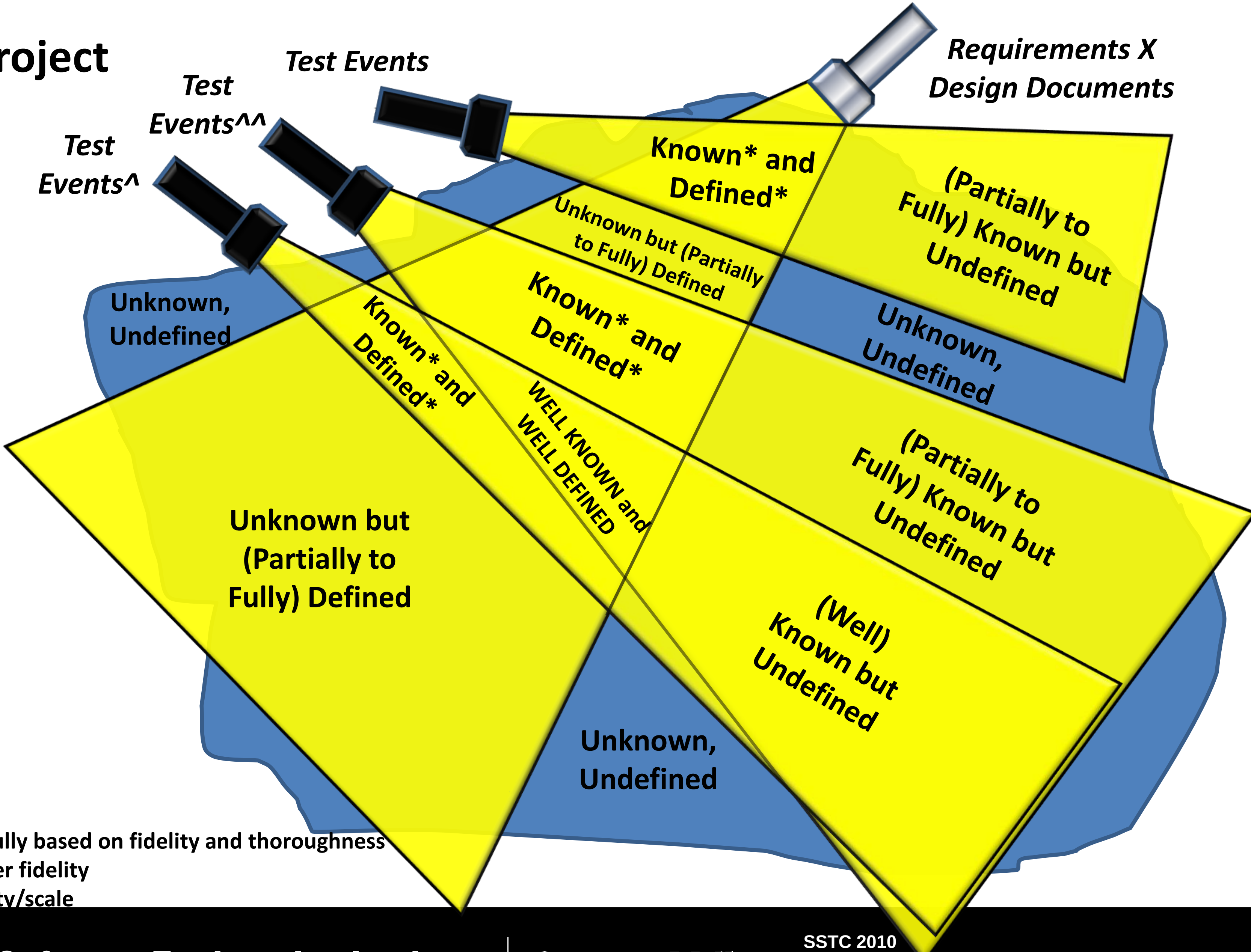
We provide a 10-step method for planning/assessing SWP, allowing for respective improvement of architecture and test processes

Our method is based on experience within a major directed SoS Service Orientated Architecture (SOA) DoD acquisition program

* See "Exploring Enterprise, System of Systems, and System and Software Architectures" by Paul C. Clements, SEI, 2009.

Illuminating the Software Performance 'Cave'

Late Project



*= Partially to Fully based on fidelity and thoroughness
^= focused, lower fidelity
^^=Higher fidelity/scale

Software Performance 10-Step Method

1 - **Make SOS SOA layout performance view**

2 - Review key resource limiters from layout

3 - Make sample scenarios: What are sources of performance impacts in each?

4 - Make list of metrics (indicate sources, architecture ties if known)

5 - Add in required SWP metrics from documents (quality/ best practice/ critical resources)

6 - Find test events that have occurred: Rate the maturity of each for each metric

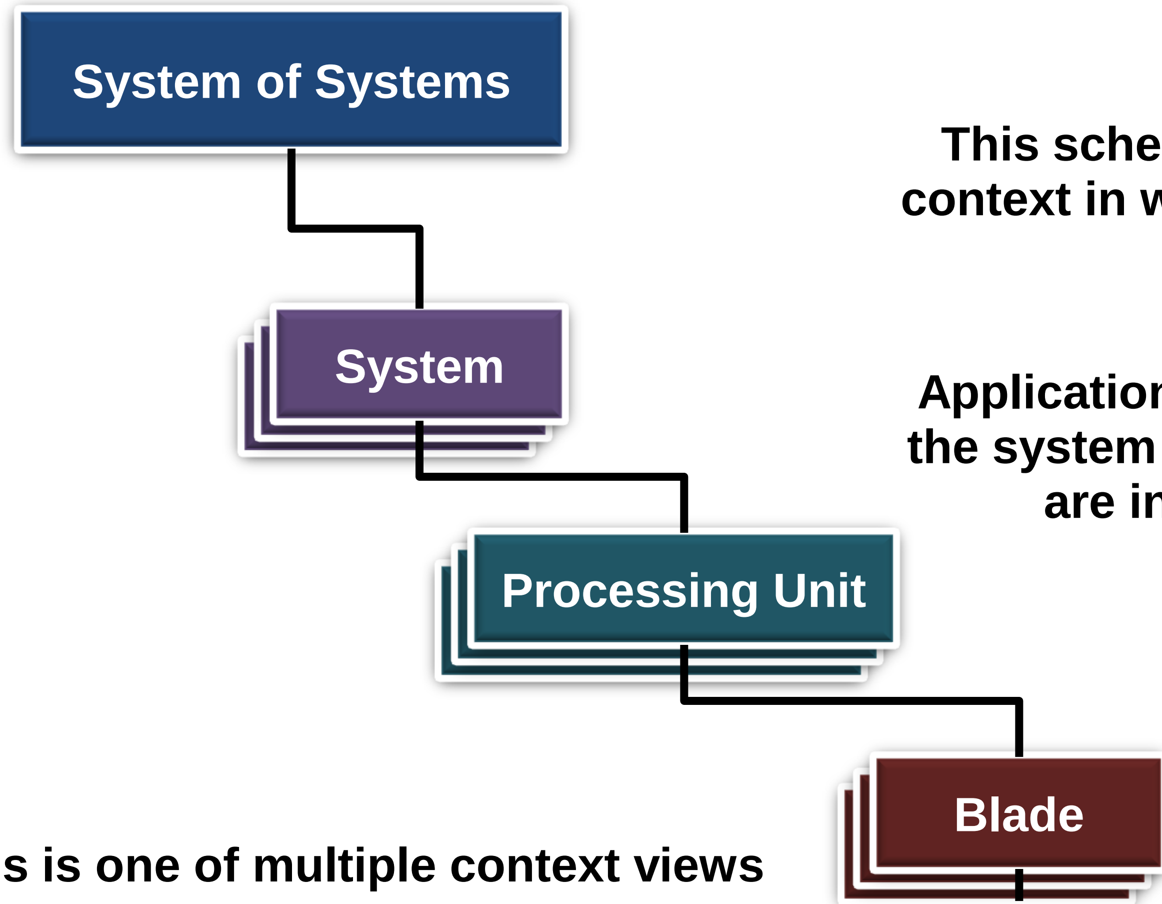
7 - Circulate results/vetting: What metrics and events are missing?

8 - Use populated metrics matrix to plan future tests and mine data from existing data sets

9 - Use architecture tie-ins to improve software performance

10 - Determine repeat schedule

An Example SoS Layout

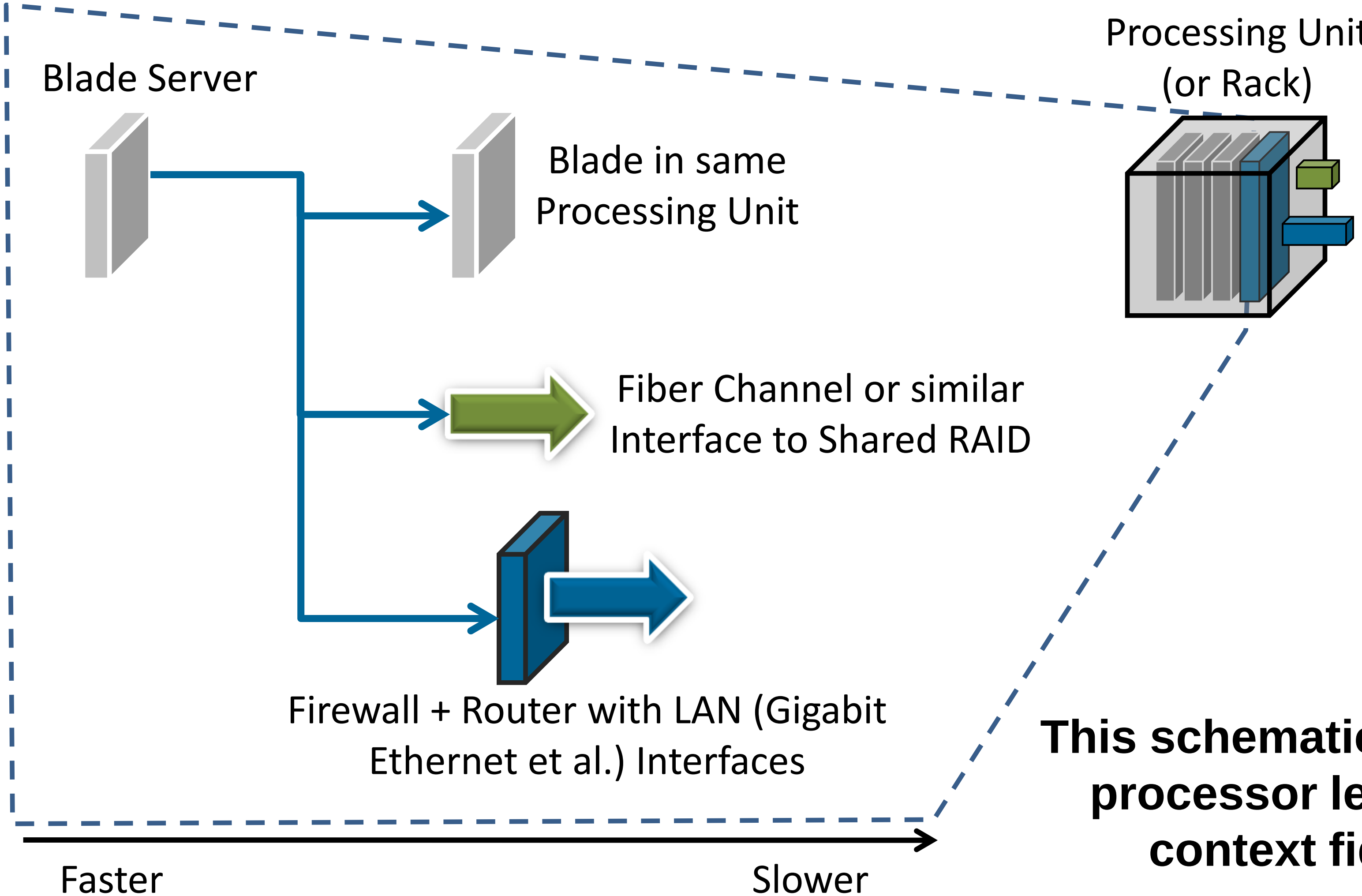


This schematic represents the SoS context in which the example software was delivered

Applications using services reside at the system level and assume services are instantiated on blades

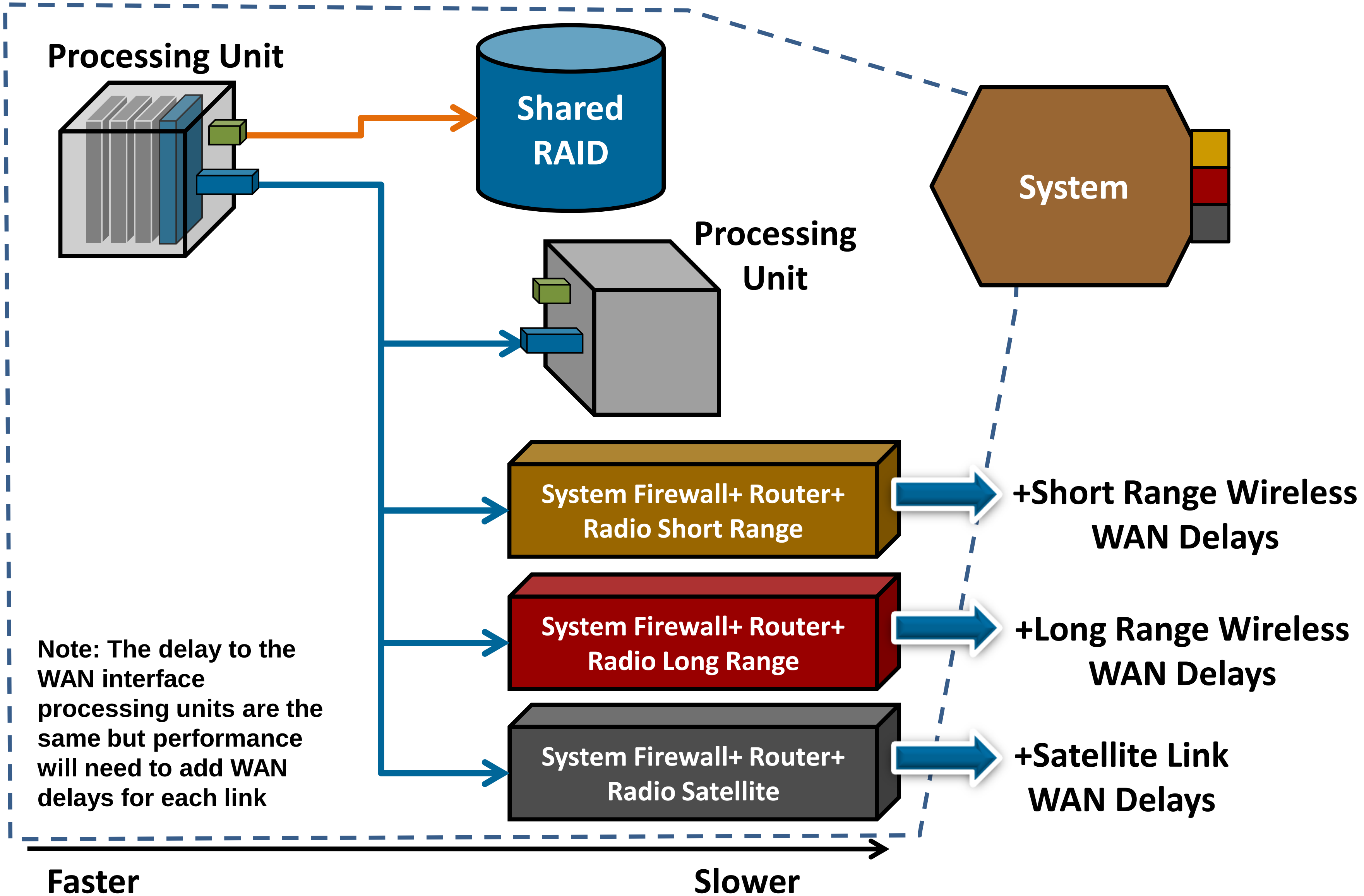
This is one of multiple context views required; it was chosen to allow further break down of performance affecting sources

Notional SoS Layout: *On a Processing Unit*



This schematic provides processor level SoS context fidelity

Notional SoS Layout: *A System*



Software Performance 10-Step Method

1 - Make SOS SOA layout performance view

2 - Review key resource limiters from layout

3 - Make sample scenarios: What are sources of performance impacts in each?

4 - Make list of metrics (indicate sources, architecture ties if known)

5 - Add in required SWP metrics from documents (quality/ best practice/ critical resources)

6 - Find test events that have occurred: Rate the maturity of each for each metric

7 - Circulate results/vetting: What metrics and events are missing?

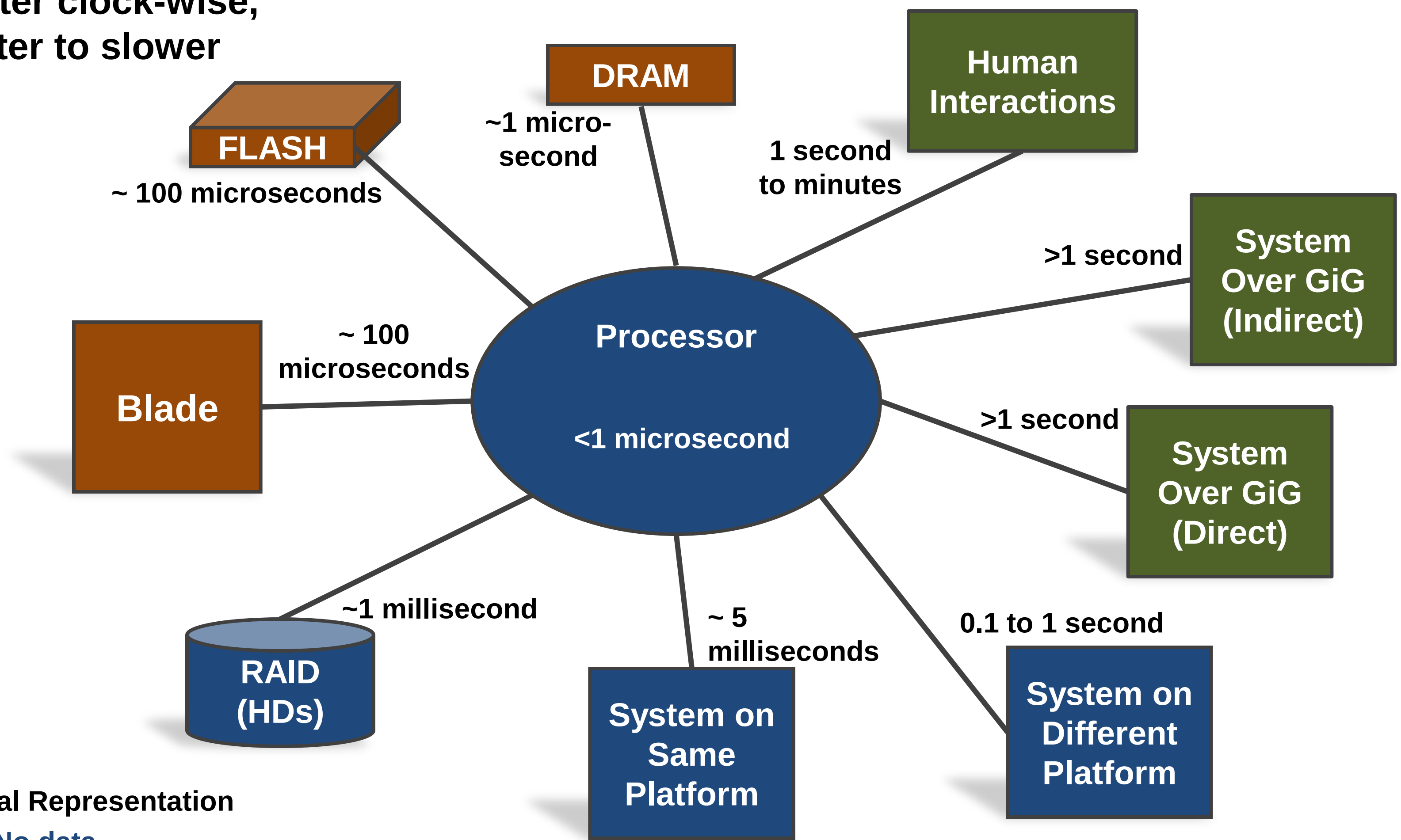
8 - Use populated metrics matrix to plan future tests and mine data from existing data sets

9 - Use architecture tie-ins to improve software performance

10 - Determine repeat schedule

Software/Hardware Performance Planning

Counter clock-wise,
faster to slower



Notional Representation
Blue =No data
Orange = Simulated Data
Green: Live Data

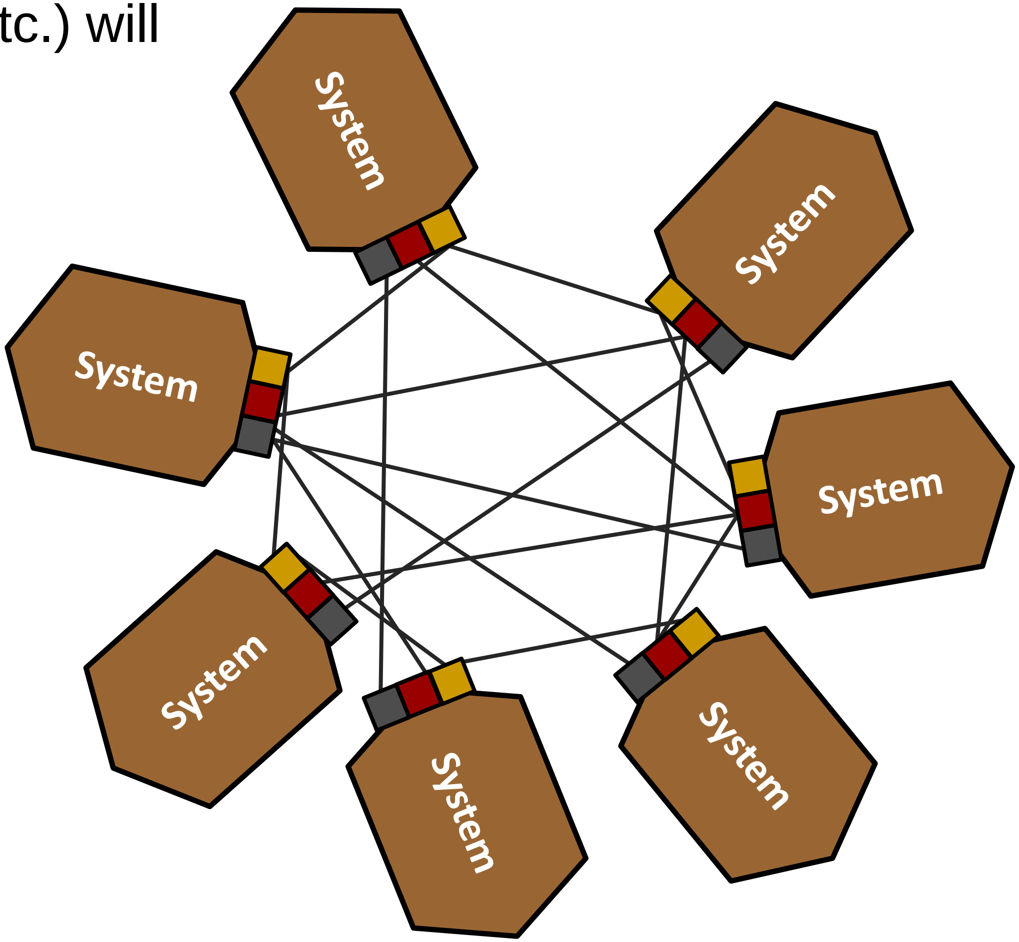
Designers should manage access to slower methods when possible

Scale Issues

The work of each blade (CPU/memory/ LAN utilization, middleware, etc.) will increase based upon

- total number of systems in the system of systems
- how often the users need services in other systems/ processing units/blades

Each increase in scale increases resource needs per service hosting blade



Software Performance 10-Step Method

1 - Make SOS SOA layout performance view

2 - Review key resource limiters from layout

3 - Make sample scenarios: What are sources of performance impacts in each?

4 - Make list of metrics (indicate sources, architecture ties if known)

5 - Add in required SWP metrics from documents (quality/ best practice/ critical resources)

6 - Find test events that have occurred: Rate the maturity of each for each metric

7 - Circulate results/vetting: What metrics and events are missing?

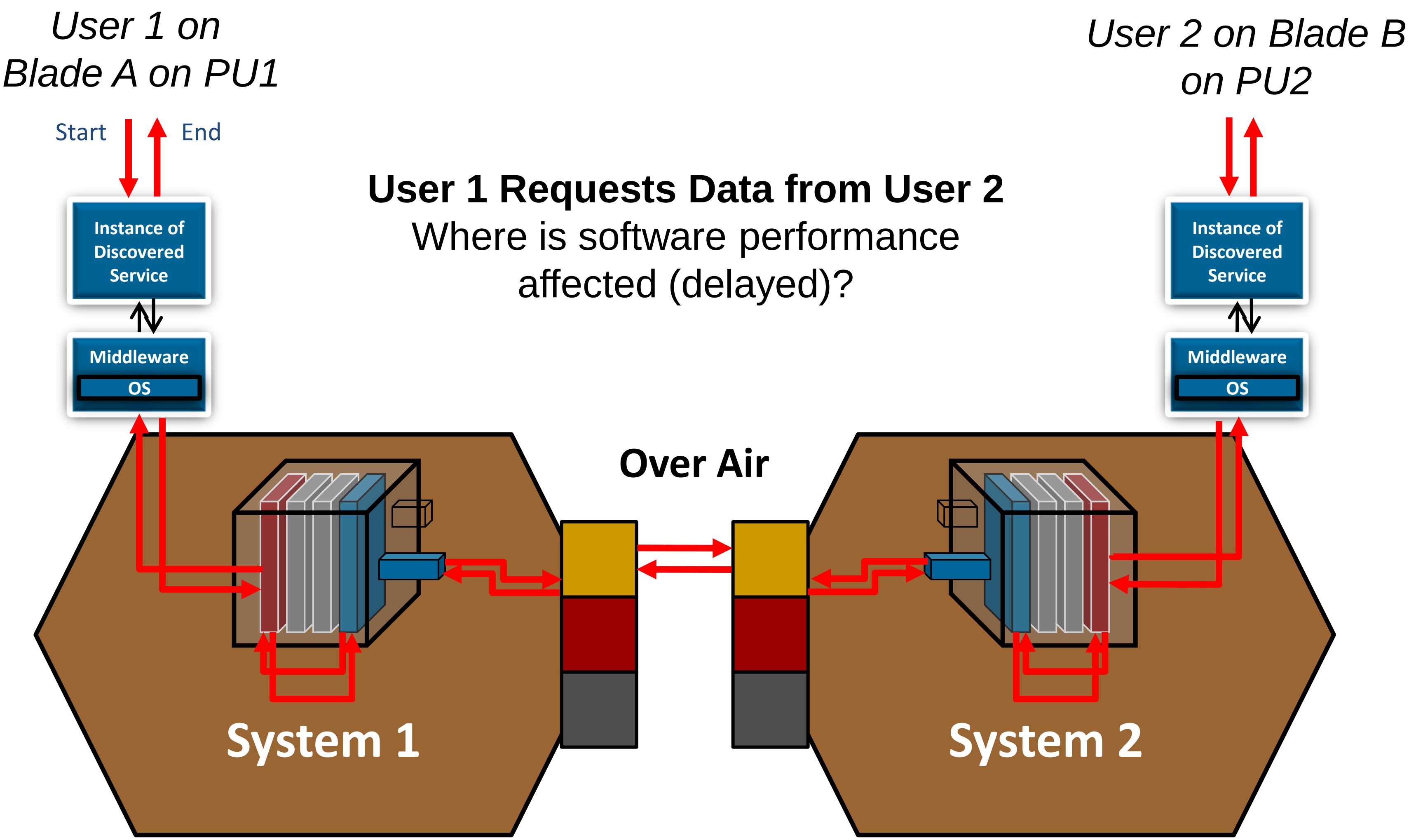
8 - Use populated metrics matrix to plan future tests and mine data from existing data sets

9 - Use architecture tie-ins to improve software performance

10 - Determine repeat schedule

A Possible Scenario - 1

User 1 Requests Data from User 2
Where is software performance affected (delayed)?



A Possible Scenario - 2

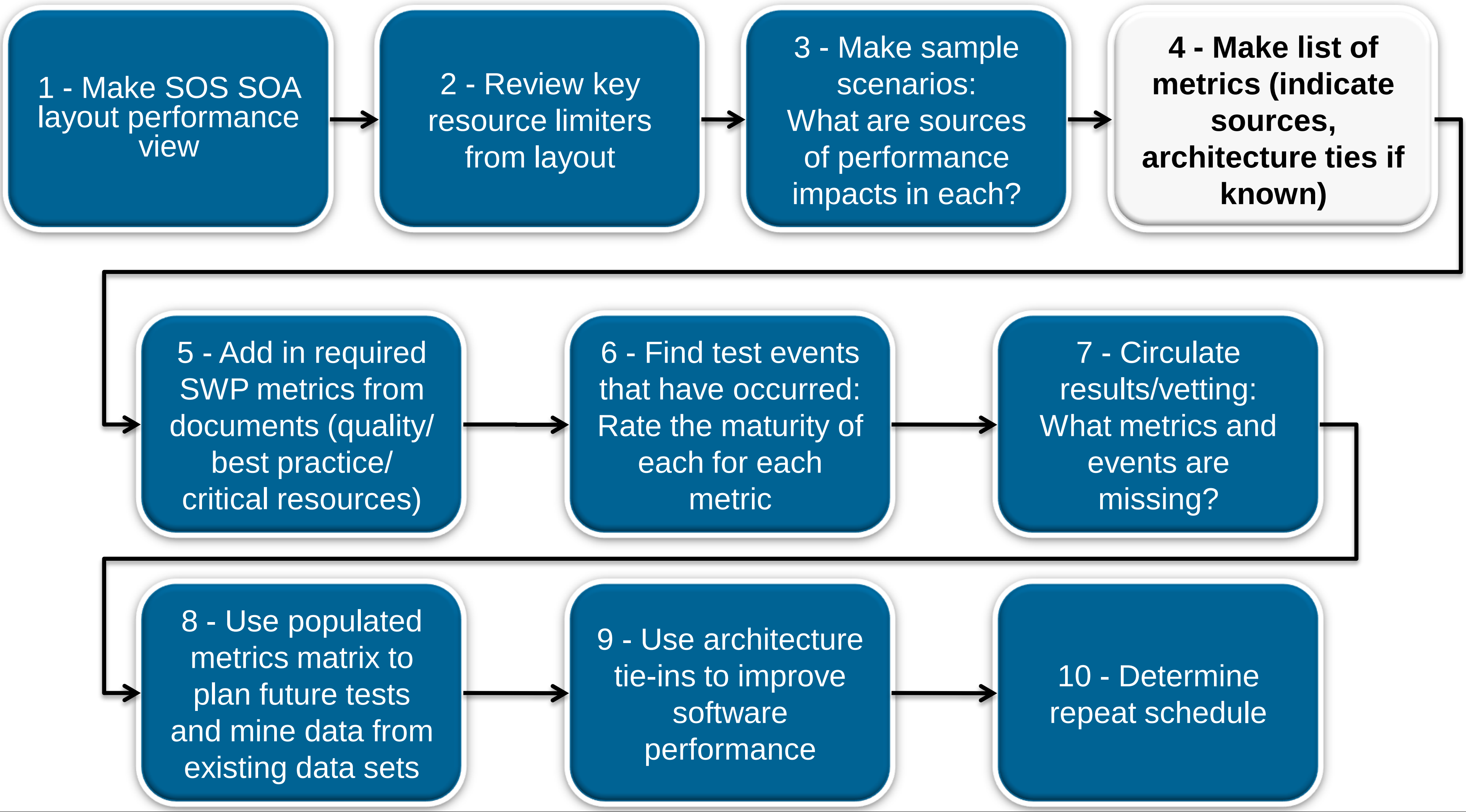
What metrics affect software performance in previous scenario?

User 1 to User 2, examples:

- On Blade A: Service Call to Middleware
- Delays Between Blade & Processing Unit
- Delays on Short Range Router/FW /Radio 1
- Delays on Short Range Router/FW /Radio 2
- LAN Latency From Short Range Router/FW/Radio 2 to PU2’s LAN Blade

User 2 to User 1: Reverse previous bullet!

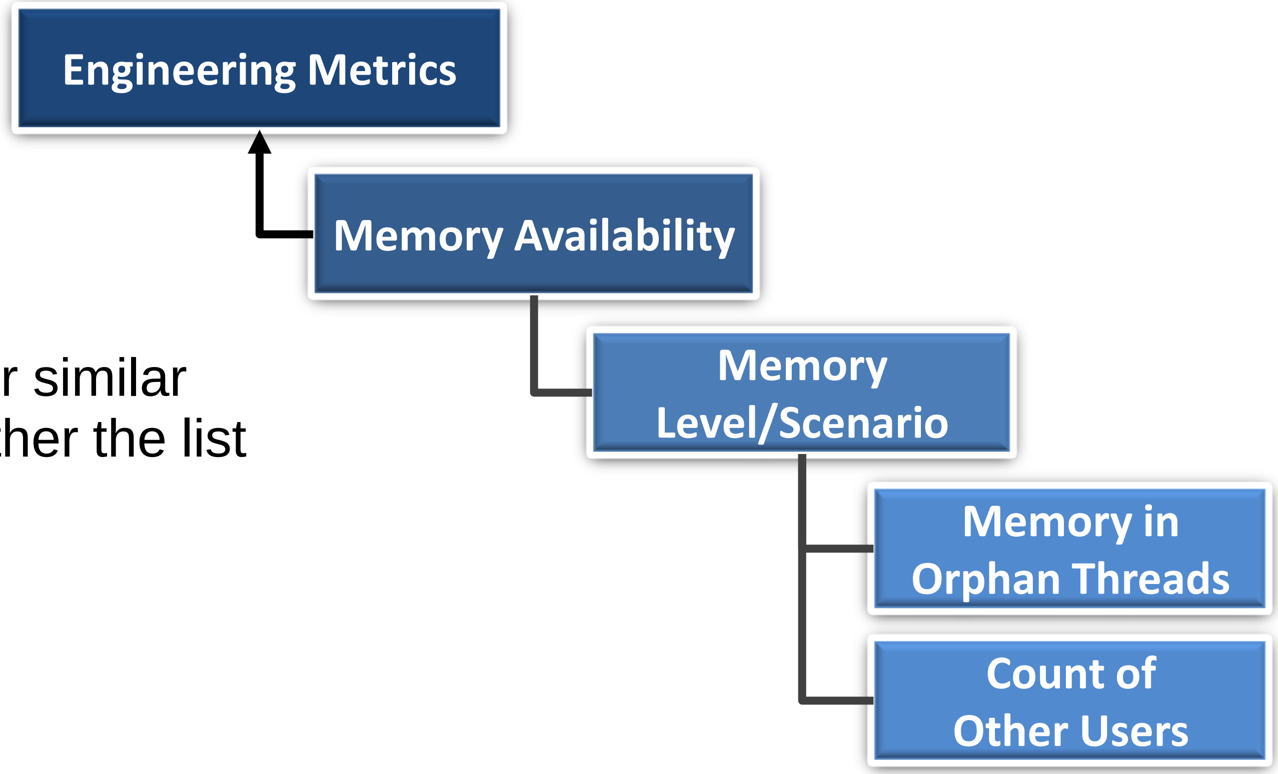
Software Performance 10-Step Method



Make a Software Performance Metrics Matrix

Consider the design levels and requirements

- Aid: ‘desk’ running scenarios
  from: intended use, take to break (‘rainy day’), and requirements



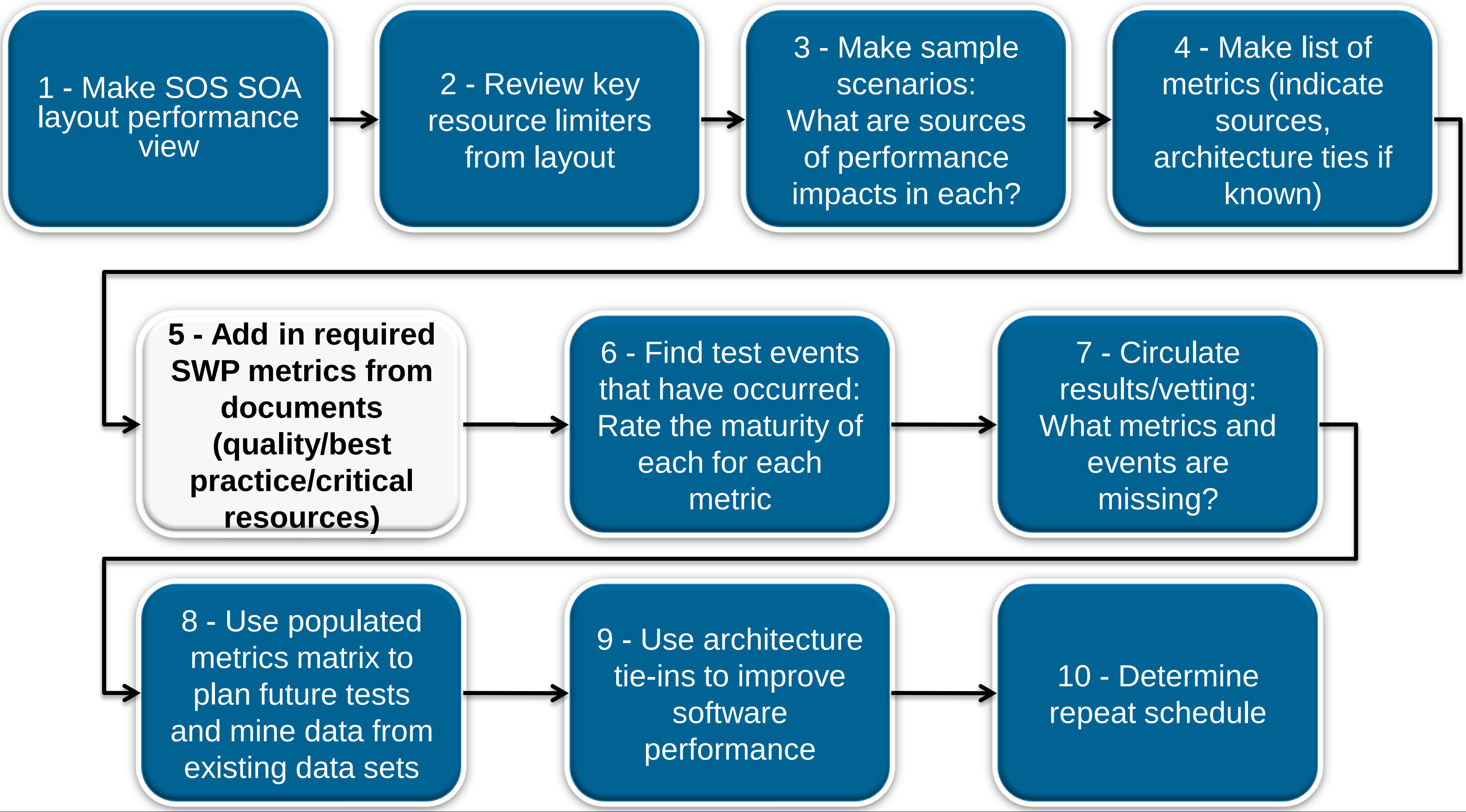
A breakout diagram or similar can be used to gather the list

The Initial Matrix

- Metric name: title, short name and key words for tagging
- Why it should be collected, including *Need Type*
- An example of the ways to collect it: *How?*
- Any ties to requirements, directly or as contributors
- *High-Level Type:* What aspect of the overall design am I assessing?

#	Short Name	Metric Title	Why?	Keywords (for Tagging)	How?	Need Type	High Level Type
1	Bcalls_ Count	Blade to blade calls (tagged by service, by process, by user, by case/scenario/time	Limiting calls from blade to blade reduces time (due to bus use)	Blade, calls, count, service, process	Bus monitoring via Processing Unit against process monitor	Efficiency	Engineer
2	HDCalls_ Count	Service traffic count to drives	Which services, applications, clients of applications are hitting the drives often. The more often RAM is used in lieu of the drives, the quicker the app will run.	User, service, raid, calls	Process-message snapshots and parse (or logging parse) for OS+bus capture (log parse)	Efficiency	Engineer

Software Performance 10-Step Method



Adding Metrics Using Existing Matrix Guidance

Use list of 20 minimums to fill in list made from scenarios

- This provides a set of metrics that might not have emerged from Step 4 scenarios, but come from experience with similar systems

Add quality metrics related to software performance

Add guidance from requirements documents

Sample Key Metrics for Software Performance				
#	Short Name	Metric Title	Why?	How?
1	HDPart_Ut	Partition/disk usage over time/scenario/ factor	Avoid overfilling partitions (which can slow or stop a system); determine which situations stress disks	Repeated capture from OS
2	LAN_Util	Platform LAN utilization	Prevent overuse of LAN on platform; watch for processes that could be done in blade instead of over LAN	SNMP MIB from routers
3	RAM_Util	RAM utilization (by client, service, application) over time	Prevent over-utilization, prevent resource hogging/application	Repeated capture from OS

Software Performance 10-Step Method

1 - Make SOS SOA layout performance view

2 - Review key resource limiters from layout

3 - Make sample scenarios: What are sources of performance impacts in each?

4 - Make list of metrics (indicate sources, architecture ties if known)

5 - Add in required SWP metrics from documents (quality/ best practice/critical resources)

6 - Find test events that have occurred: Rate the maturity of each for each metric

7 - Circulate results/vetting: What metrics and events are missing?

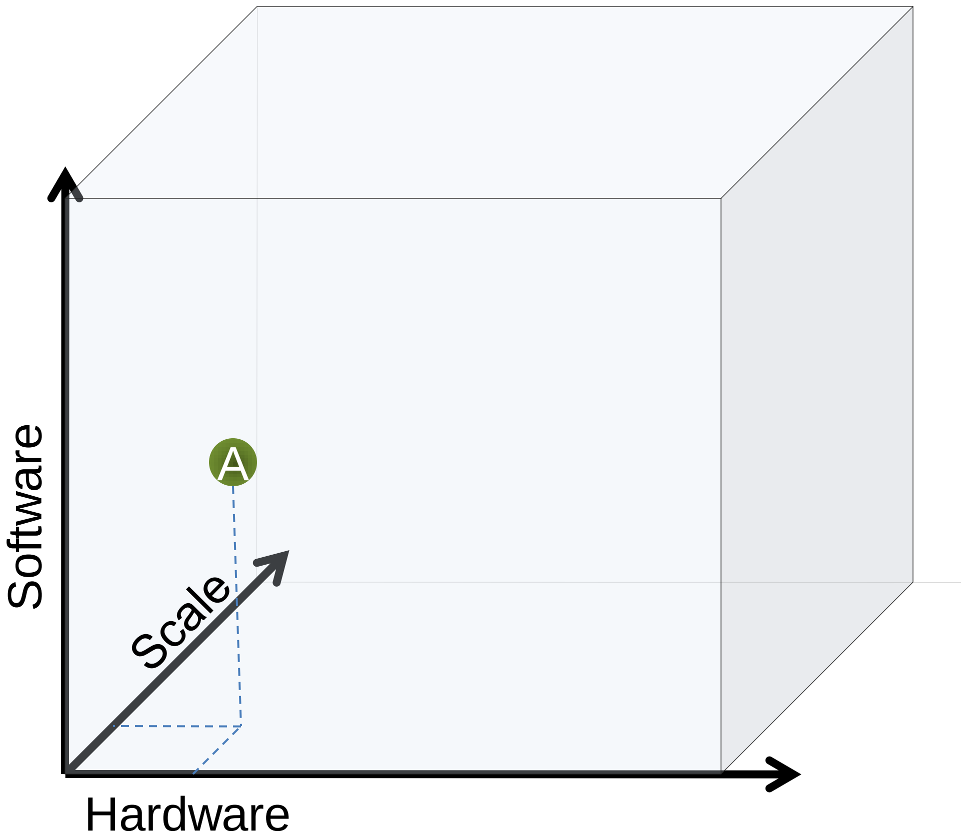
8 - Use populated metrics matrix to plan future tests and mine data from existing data sets

9 - Use architecture tie-ins to improve software performance

10 - Determine repeat schedule

Testing/Simulation Types

Cube of 'Realism' (Omitting Network*)



* One could extend to 'Network' for a 4th Dimension

Assess realism per test event

1. Software
- Mod=Modeled
- Sim=Simulated
- Proto=Prototype
- EB=Early Build
- LB=Later Build
- Mat=Mature

2. Hardware
- Sim=Simulated
- EP=Early Prototype
- LP=Late Prototype
- IP=Initial Production
- FP=Full Production

3. Scale
- SB/MB=Single Blade/Multiple Blades
- PU/MPU=Process Unit/Multiple PUs
- SS=Single System
- LS=Limited Multiple System
- PS=Partial Scale
- FS=Full Scale

Test for Realism

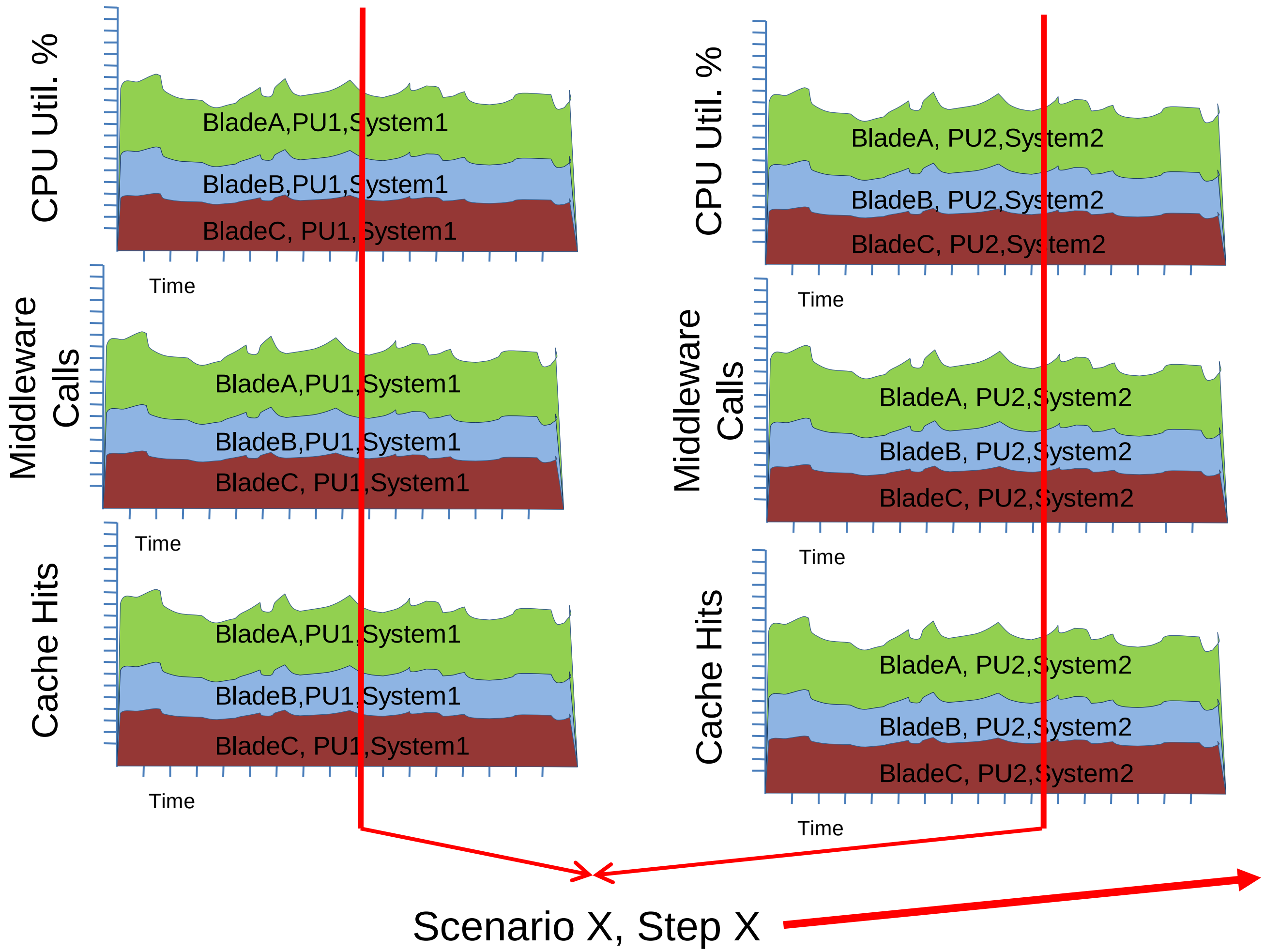
Realism varies by metric inside each test event due to available test assets and timeframes

Test targeted at reducing one set of risks might collect data on other related areas as a side effect

Review of full test artifacts can mine for ‘off-target’ collections

Off-target metric collections might be at a lower fidelity level than metric included in risk target of test

Trending and Correlation



Other correlations

- Regression comparisons?
- Gap analysis; compare w/desired performance

Tie to architecture (design, various levels)

System Architecture; Software Architecture

Which cross correlations have a payoff?

Software Performance 10-Step Method

1 - Make SOS SOA layout performance view

2 - Review key resource limiters from layout

3 - Make sample scenarios: What are sources of performance impacts in each?

4 - Make list of metrics (indicate sources, architecture ties if known)

5 - Add in required SWP metrics from documents (quality/ best practice/critical resources)

6 - Find test events that have occurred: Rate the maturity of each for each metric

7 - Circulate results/vetting: What metrics and events are missing?

8 - Use populated metrics matrix to plan future tests and mine data from existing data sets

9 - Use architecture tie-ins to improve software performance

10 - Determine repeat schedule

Who Vets the SWP Metrics Matrix?

Testing groups are usually scattered in various system groups and at program level

Bring representatives of each group together to examine each iteration of metrics matrix

- **Limit** attendance to those who understand test metrics and fidelity levels
- Honesty, not spin, is important
- Get leadership backing

Vet matrix with this newly-formed *Technology Interchange Group* (TIG).

Vetted Matrix and Procedure Linkage

Use matrix as a starting point for discussion for initial TIG meeting

- Discuss matrix data: Was anything missed?
  - All that has happened to date: *Does it include all test events?*
  - Knowledge of events at each scale: *Does it capture the correct realism and scale of each event?*
- Revise matrix
  - Include missed or incomplete items discovered
  - Gain consensus on correctness/completeness of metrics: *Are we measuring the right performance? Does the list account for SWP issues that may emerge later?*

Re-circulate to confirm results

- Store matrix in configuration-controlled, commonly accessible location (Sharepoint, Wiki, etc.)
- Encourage TIG to comment and distribute to their teams for comment
- Collect comments, confirm veracity of updates with TIG, revise matrix

Repeat until there is a strong confidence/consensus in matrix

Software Performance 10-Step Method

1 - Make SOS SOA layout performance view

2 - Review key resource limiters from layout

3 - Make sample scenarios: What are sources of performance impacts in each?

4 - Make list of metrics (indicate sources, architecture ties if known)

5 - Add in required SWP metrics from documents (quality/ best practice/critical resources)

6 - Find test events that have occurred: Rate the maturity of each for each metric

7 - Circulate results/vetting: What metrics and events are missing?

8 - Use populated metrics matrix to plan future tests & mine data from existing data sets

9 - Use architecture tie-ins to improve software performance

10 - Determine repeat schedule

Metrics/Planning for Metrics Collection using the Matrix

Insert metrics with low event coverage into future test events.

- What metrics (rows) in the matrix have no associated events (i.e. empty columns)? Which metrics were only measured at a low scale or fidelity?
- Insert metrics into event plans and insert planned events into the matrix

Make metric list a standard minimum for tests at any scale

Create correlation standards and a history of what correlations have lead to problem discovery

Agree on initial conditions for tests

Ideas for Entry Criteria: *Metrics Infrastructure*

Consolidated *Metrics Library Database*

- Complex trends and simple points
- Easily accessible by architects/engineers/development/other test groups
- Metadata tagging using a standard

Insert into test schedule

- Run future test event planning through TIG
- Invite group edits to matrix

Software Performance 10-Step Method

1 - Make SOS SOA layout performance view

2 - Review key resource limiters from layout

3 - Make sample scenarios: What are sources of performance impacts in each?

4 - Make list of metrics (indicate sources, architecture ties if known)

5 - Add in required SWP metrics from documents (quality/ best practice/critical resources)

6 - Find test events that have occurred: Rate the maturity of each for each metric

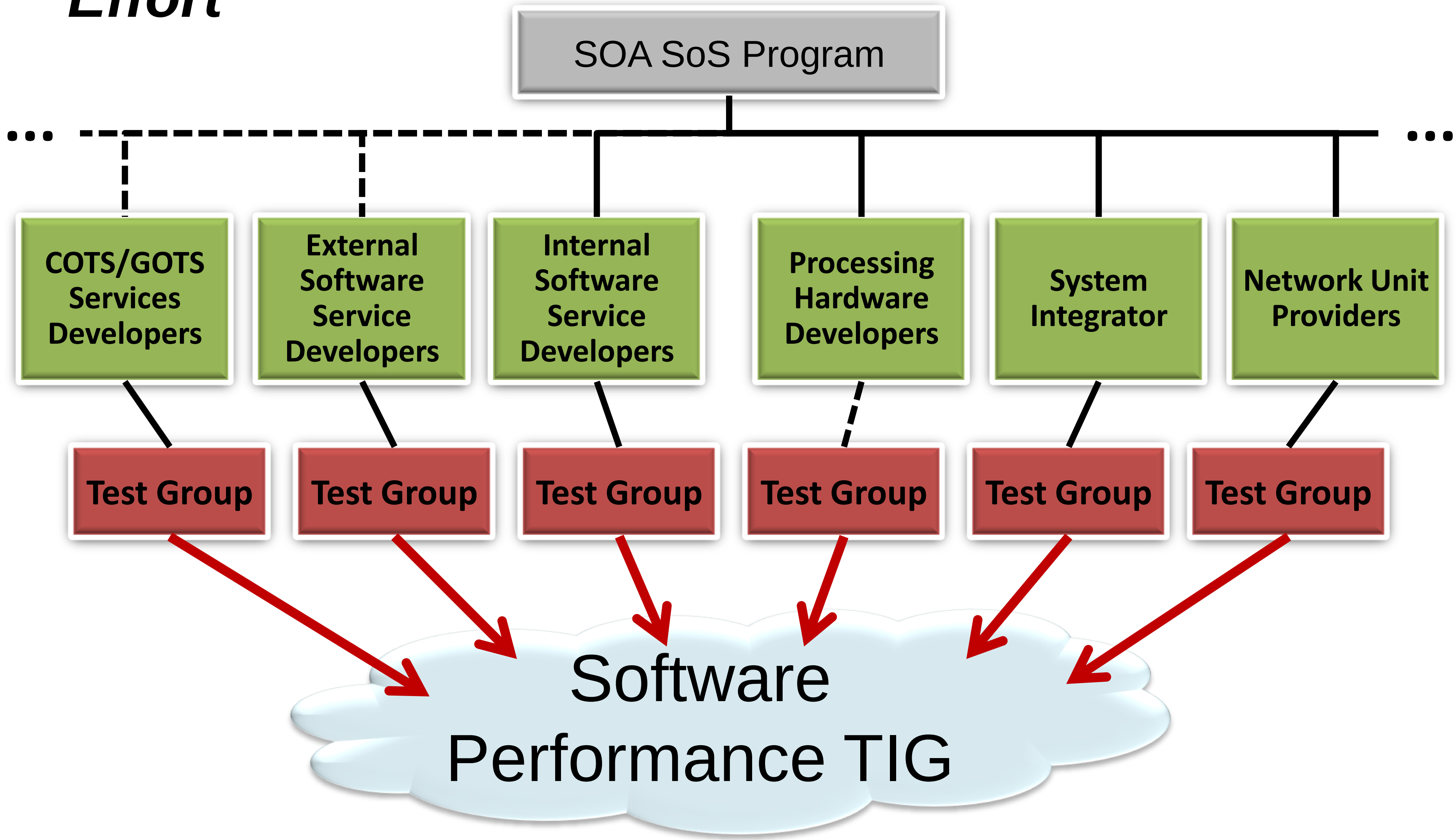
7 - Circulate results/vetting: What metrics and events are missing?

8 - Use populated metrics matrix to plan future tests & mine data from existing data sets

9 - Use architecture tie-ins to improve software performance

10 - Determine repeat schedule

Software Performance Management: *A Team Effort*



Relating Architecture to Metrics

It is useful with a vetted metrics matrix to tie each metric to architecture

- Use ties to improve performance

There are likely no orphan metrics; they are just more complex to trace to architecture and design

Repeated columns of higher fidelity and realistic events improve confidence that the metric is covered and performance quantified; use these to plan tests

Architecture and design elements tied to performance will gain confidence with successive events; again test planning

Conclusions

Understanding software performance for a SoS SOA system is complex; managers need to:

- Understand the system's respective performance affecting levels
- Develop a metrics list derived from scenarios and other sources
- Tie in test events to make the metrics matrix
- Have a way to circulate the matrix by understanding the organization
- Feedback the matrix and metrics testing results to architecture leads
- Keep the matrix current or status will be unknown

BACK-UP SLIDES

Acronym List

CM	Configuration Management
COTS	Common Off The Shelf
CPU	Central Processing Unit
DoD	U.S. Department of Defense
DRAM	Dynamic Random Access Memory
E2E	End-to-End
FW	Fire Wall
GiG	Global Information Grid
GUI	Graphical User Interface
HD	Hard Drives
H/W	Hardware
LAN	Local Area Network
LUT	Limited User Test
IPT	Integrated Process Team
M&S	Modeling and Simulation

Acronym List

OS	Operating System
PU	Processing Unit
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
RFP	Request For Proposal
SE	Systems Engineering
SEC	Army Software Engineering Center
SOA	Service Oriented Architecture
SoS	System of Systems
SW	Software
SWP	Software Performance
TIG	Technology Interchange Group
TRL	Technical Readiness Level
WAN	Wide Area Network

Services

"Services and applications are defined as primarily software based components which perform specific functions using standard interfaces. A service is defined as a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description (reference w). A service is a function that is well-defined, self contained, and does not depend on the context or state of other services. It easily allows for reuse in yet to be determined functions. Applications are designed to perform a specific function directly for the user or for another application."

US DoD CJCSI 6212.01E, 15 December 2008

System of Systems:

See “Exploring Enterprise, System of Systems, and System and Software Architectures” by Paul C. Clements, SEI: **http://www.sei.cmu.edu/library/abstracts/presentations/22jan2009webinar.cfm**

“**System of Systems (SoS) Architecture**

- A SoS is a set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities.
- Varieties:
  - **Directed: SoS objectives, management, funding and authority in place; systems are subordinated to the SoS**
  - **Acknowledged: SoS objectives, management, funding and authority in place; systems retain their own management, funding and authority in parallel with the SoS**
  - **Collaborative: No objectives, management, authority, responsibility, or funding at the SoS level; systems voluntarily work together to address shared or common interest**
  - **Virtual: Like collaborative, but systems don’t know about each other (for example, the Internet)”**

Contact Information

Presenter / Point of Contact

Jim Wessel / Bryce Meyer

Acquisition Support Program

Telephone: +1 908-418-0323 / +1 314-800-3159

Email: jwessel@sei.cmu.edu / bmeyer@sei.cmu.edu

World Wide Web:

www.sei.cmu.edu

www.sei.cmu.edu/contact.html

U.S. mail:

Software Engineering Institute

Customer Relations

4500 Fifth Avenue

Pittsburgh, PA 15213-2612

USA

Customer Relations

Email: customer-relations@sei.cmu.edu

Telephone: +1 412-268-5800

SEI Phone: +1 412-268-5800

SEI Fax: +1 412-268-6257