Exploiting Global View for Resilience (GVR)

An Outside-In Approach to Resilience

Andrew A. Chien

X-stack PI Meeting

September 18-19, 2012

Project Team

- University of Chicago: Andrew A. Chien (PI), Hajime Fujita, Zachary Rubenstein, Guoming Lu
- Argonne: Pavan Balaji (co-PI), James Dinan, Pete Beckman, Kamil Iskra
- HP Labs: Robert Schreiber
- Application Partnerships
  - Advanced Nuclear Reactor Simulation (Andrew Siegel, CESAR)
  - Computational Chemistry (Jeff Hammond, ALCF)
  - Rich Computational Frameworks (Mike Heroux, Sandia)
  - ... and more!...

THE UNIVERSITY OF CHICAGO

Argonne NATIONAL LABORATORY

Outline

- Resilience – the Challenge
- Global View Resilience (GVR) Approach
- GVR Research Challenges
- Expected Outcomes and X-stack Synergy
- Questions

Resilience Challenges

- Can we achieve a smooth transition to system resilience? (a la Flash memory, Internet)
- What's an application to do?

Resilience Co-design

- Software: Information and Algorithms to enhance resilience (REQ: Portable, flexible)
- Runtime, OS, and Architecture Mechanisms to enhance resilience (REQ: leverage beyond HPC, cheap)

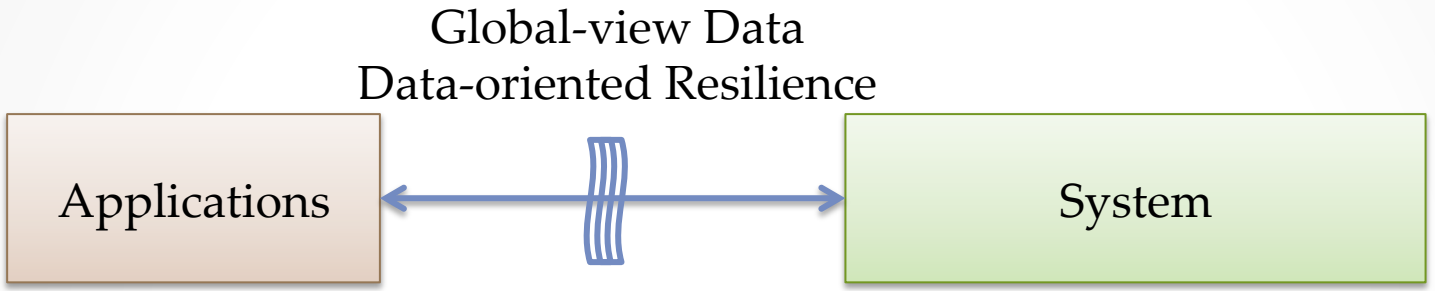
"Co-design without co-dependence"

Challenge

- Enable an application to incorporate resilience incrementally, expressing resilience proportionally to the application need

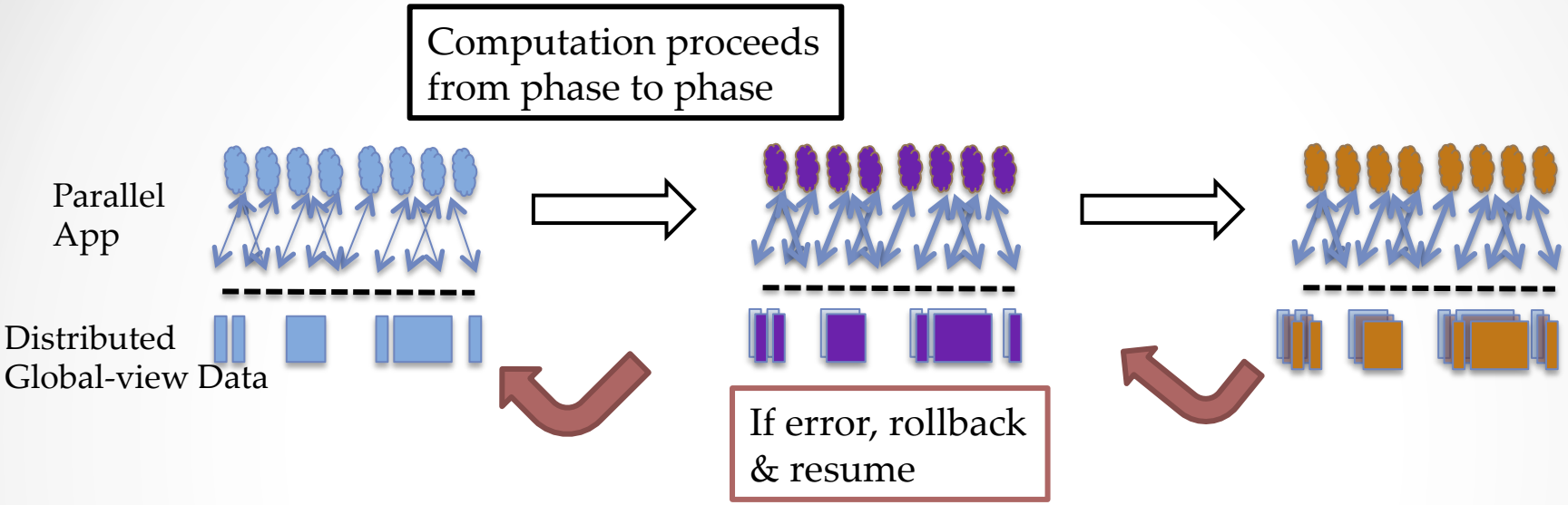
- "Outside in", as needed, incremental, ...

GVR Approach



- Application-System Partnership
  - Expose and exploit algorithm and application domain knowledge
  - Enable "End to end" resilience model
- Foundation in Data-oriented resilience
  - Internet services, map-reduce, internet, ...
  - Achieve with high performance and massive parallelism...
  - Global view data Foundation (PGAS..., GA, Swarm, Parallex,CnC, ...)

Data-oriented Resilience

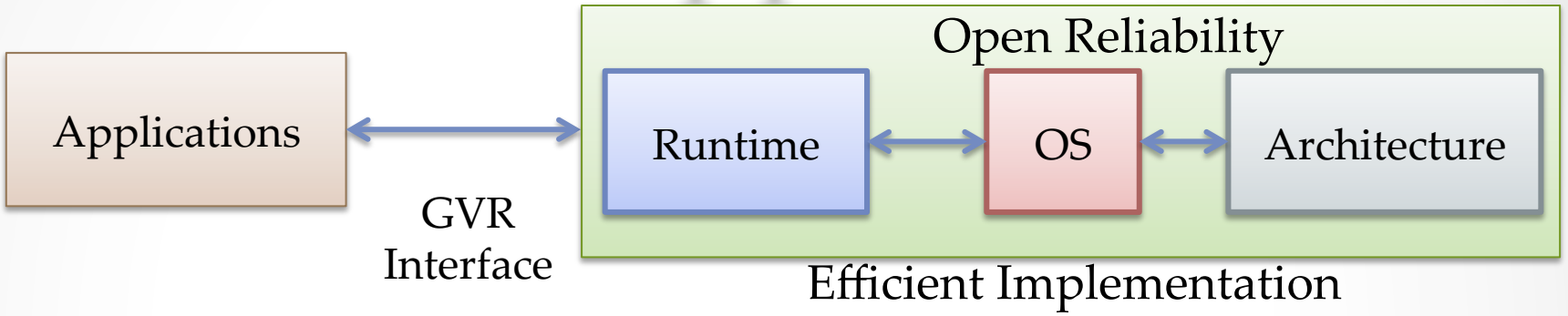


- Parallel applications and global-view data
- Natural parallel structure version-to-version
  - Example: shock hydro simulation at t=10ms to 100ms
  - Example: iterative solver at iteration 1 to 20
  - Example: monte carlo at 10M to 20M points
- Temporal redundancy enables rollback and resume
  - User-controlled, convenient

Resilience Partnership

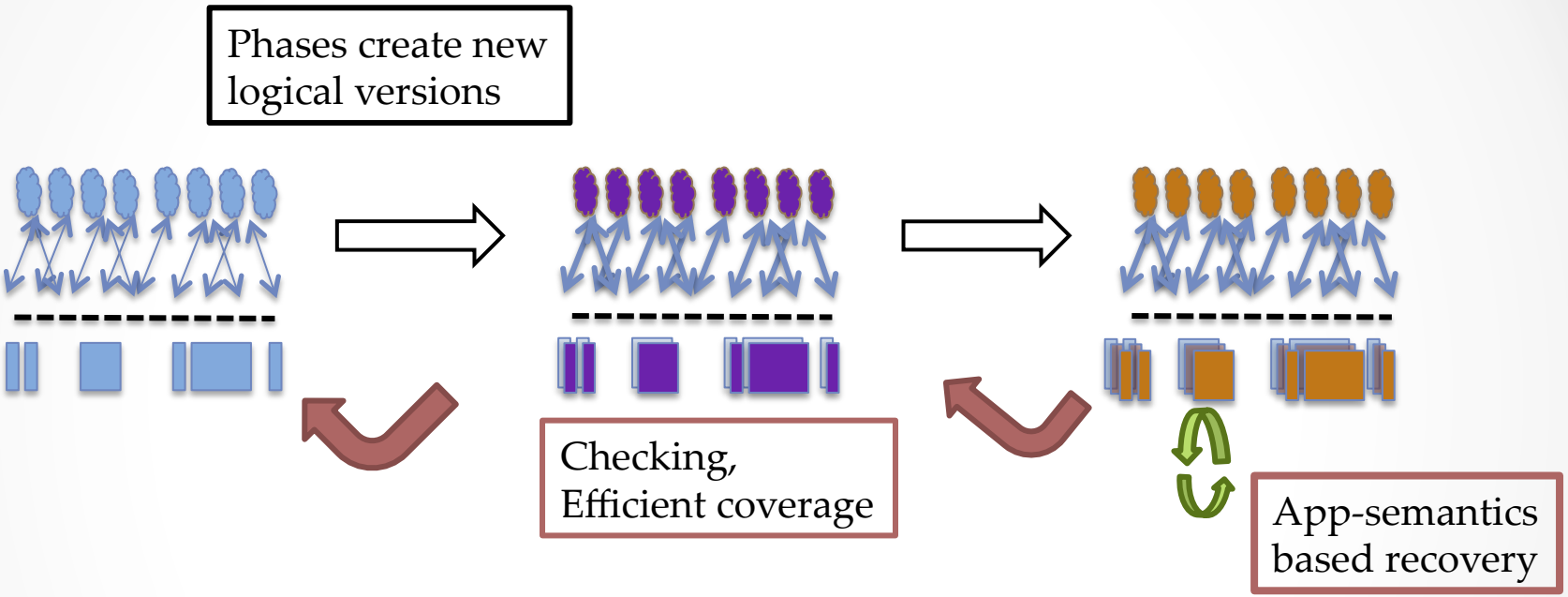
- Proportional Resilience
  - Application specifies "Resilience priorities"
  - Mapped into data-redundancy in space
  - Mapped into redundancy in time (multi-version)
  - *Complements computation/task redundancy efforts*
- Deep error detection: invariants, assertions, checks ... and recovery
  - Applications add further checks based on algorithm and domain semantics
  - Application add flexible, adaptive recovery mechanisms (and exploit multi-version)
- "End-to-end" resilience

GVR Approach



- x-layer approach for efficient execution (and better resilience)
  - Spatial redudancy – coding at multiple levels, system level checking
  - Temporal redundancy -- Multi-version memory, integrated memory and NVRAM management
- Push checks to most efficient level (find early, contain, reduce ovhd)
- Recover based on semantics from any level (repair more, larger feasible computation, reduce ovhd)
- Efficient implementation support in runtime, OS, architecture … increase efficiency and containment

Multi-version Memory



- Common parallel paradigm, basis for programmer engagement
- Frames invariant checks, more complex checks based on high-level semantics
- Frames sophisticated recovery

Research Challenges

- Understand application resilience needs and opportunities for "proportional resilience" and "deep error detection"/ "end-to-end resilience"
- Explore multi-version memory as opportunity for framing richer resilience and parallelism
- Design API that embodies these ideas and "gentle slope" incremental application effort
- Create efficient x-layer implementations – many questions
- Explore architecture opportunities to increase resilience and reduce overhead

Global-view data Program

```
GV_Allocate(gv_a,gv_b,gv_c);
GV_Distribution(gv_c, me, lo, hi);
While (not_converged()){  // ... Loop until done...
     Get(gv_a, lo2, hi2, a, ld);
     Get(gv_b, lo3, hi3, b, ld);
     GV_sync();
     for(i=0; i < hi3[0]-lo3[0]+1; i++)
       for(j=0; j < hi3[1]-lo3[1]+1; j++)
          btrns[j][i] = b[i][j];
     for(i=0; i < hi[0] - lo[0] + 1; i++) {
       for(j=0; j < hi[1] - lo[1] + 1; j++) {
         c[i][j] = 0.0;
         for(k=0; k<dims[0]; k++)
           c[i][j] = c[i][j] + a[i][k]*btrns[j][k];
       }
     }
     Put(gv_c, lo, hi, c, ld);
     GV_sync();
}
GV_destroy(gv_a,gv_b,gv_c);
```

Create and Cleanup

Get and Put Data

Synch Operations

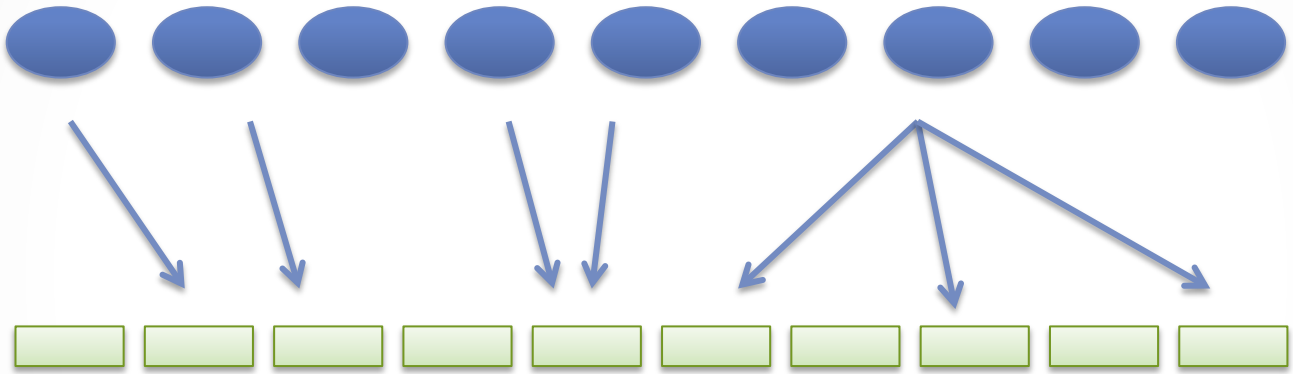
GVR Resilient Program

```
GV_Allocate(gv_a,gv_b,gv_c);        ➔  GVR_allocate(gv_a,gv_b,HIGH);
GV_Distribution(gv_c, me, lo, hi)➔  GVR_allocate(gv_c);          .
While (not_converged()){  // ... Loop until done...
     Get(gv_a, lo2, hi2, a, ld);
     Get(gv_b, lo3, hi3, b, ld);
     GV_sync(); ➔   GVR_version_inc(gv_a,gv_b,gv_c);
     for(i=0; i < hi3[0]-lo3[0]+1; i++)
       for(j=0; j < hi3[1]-lo3[1]+1; j++)
          btrns[j][i] = b[i][j];
     for(i=0; i < hi[0] - lo[0] + 1; i++) {
       for(j=0; j < hi[1] - lo[1] + 1; j++) {
         c[i][j] = 0.0;
         for(k=0; k<dims[0]; k++)
           c[i][j] = c[i][j] + a[i][k]*btrns[j][k];
       }
     }
     Put(gv_c, lo, hi, c, ld);
     GV_sync(); ➔   GVR_sync(gv_a,gv_b,gv_c);
}
GVR_destroy(gv_a,gv_b,gv_c);
```

Resilience Priorities

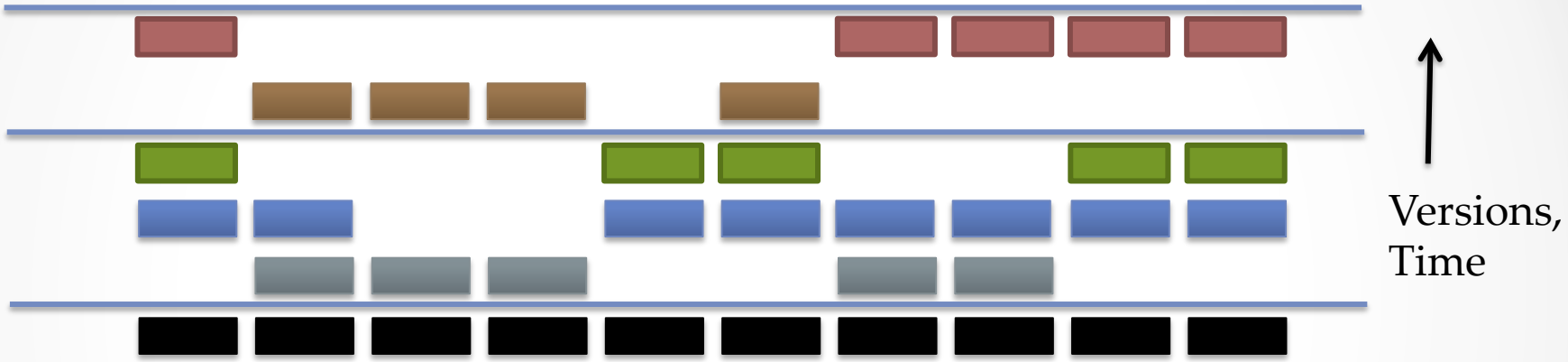
Version-Syn Operations

Global view & Consistent Snapshots



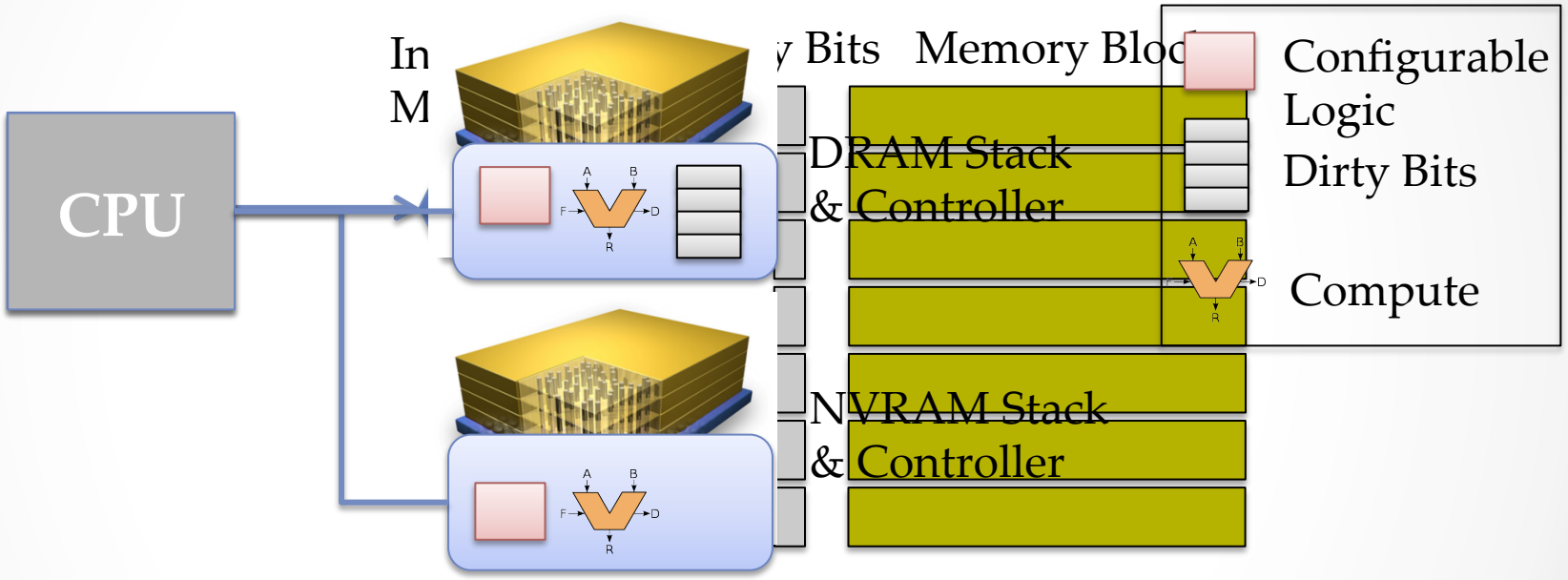
- How to safely, efficiently identify consistent snapshots?
  - Application control: Global Synch; Array-level synch; explicit snapshot
  - Application flagged (optional)
  - Implicit (runtime decides)
- Snapshots = natural points to express and implement assertions, checks, recovery

Implementing Multi-version



- How to implement multi-version efficiently?
  - Time, Space, Label => representation, protocol
- Which to take?
  - Versions are logical, snapshots require resources
- Intelligent storage:
  - Representation, compression, architecture support
  - Older versions recede into storage [SILT]

Intelligent Memory and Storage



- How to exploit intelligence at memory and storage? (at controller)
- Intelligent stacked DRAM and storage-class Memory [HMC,PIM]
- Fine-grained state tracking; compression, intelligent, copying, etc.
- Efficient version capture; differenced checkpoints (Plank95, Svard11)

Resilience

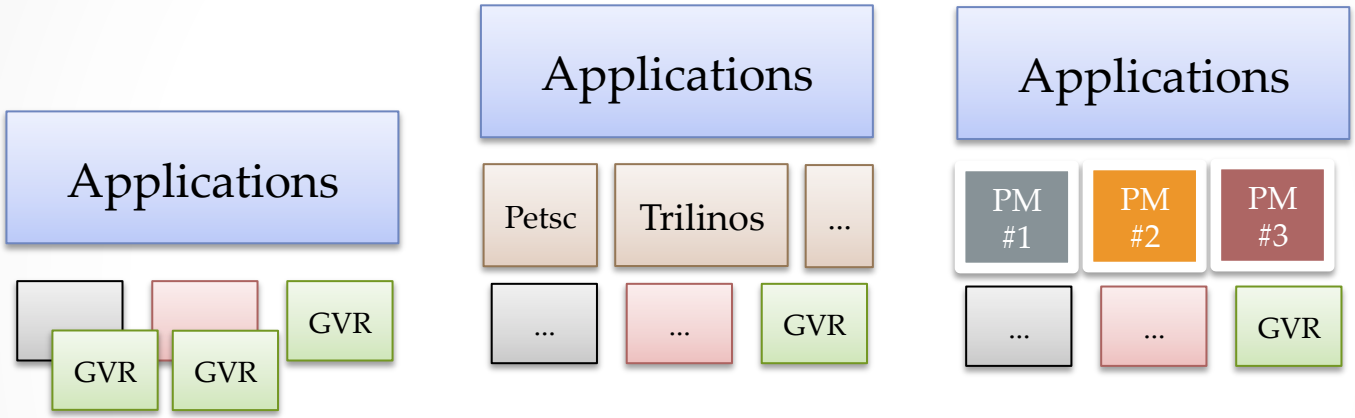
- Resilience is like "virtue", everyone wants it, but no one wants to pay for it.

- Opportunities:
  - Multi-version and increased concurrency
  - Multi-version and debugging
  - Architecture support and fine-grained synchronization, application checks, compressed memory, etc.
  - ...more?

Expected Outcomes

- Use cases – Application skeleton design and classifications which form foundation of the design
- Design of GVR API for flexible resilience and multi-version global data
- Research prototype software developed as a library; target for programmers, compiler backends
- Experiments with mini-apps and application partners (w/ co-design postdocs)
- Assessment of architecture support opportunities and quantitative benefits

GVR X-stack Synergies



- Direct Application Programming Interface
- Co-existence, even target with other Runtimes
- Rich Solver Library Building Block
- Programming System Target

Discussion

- Eager for x-stack collaborators
- Applications, programming systems, libraries

- Questions?