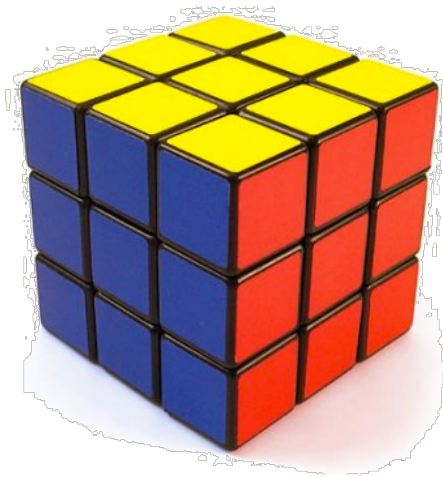


Lind

Challenges turning virtual composition into reality



Chris Matthews

Justin Cappos

Rick McGeer

Stephen Neville

Yvonne Coady

University of Victoria

NYU-Poly

HP Labs

University of Victoria

University of Victoria

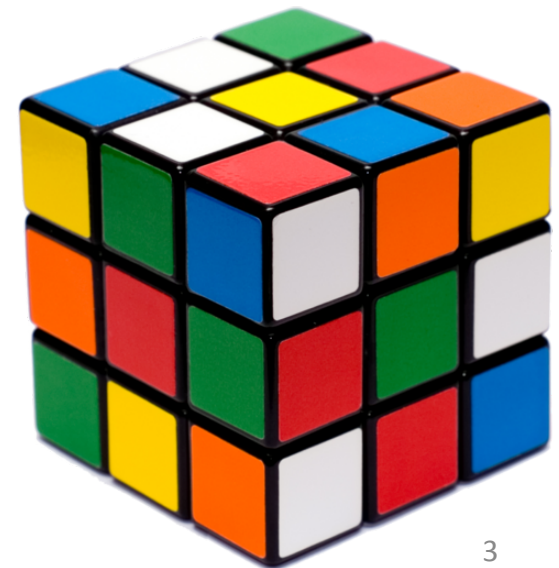
Outline

- Talk about useful isolation mechanisms
- Virtual Components
 - **Secure**
 - Fault Tolerant
 - Dynamic
 - Scalable
- Implementation
- Evaluation techniques
- Discussion

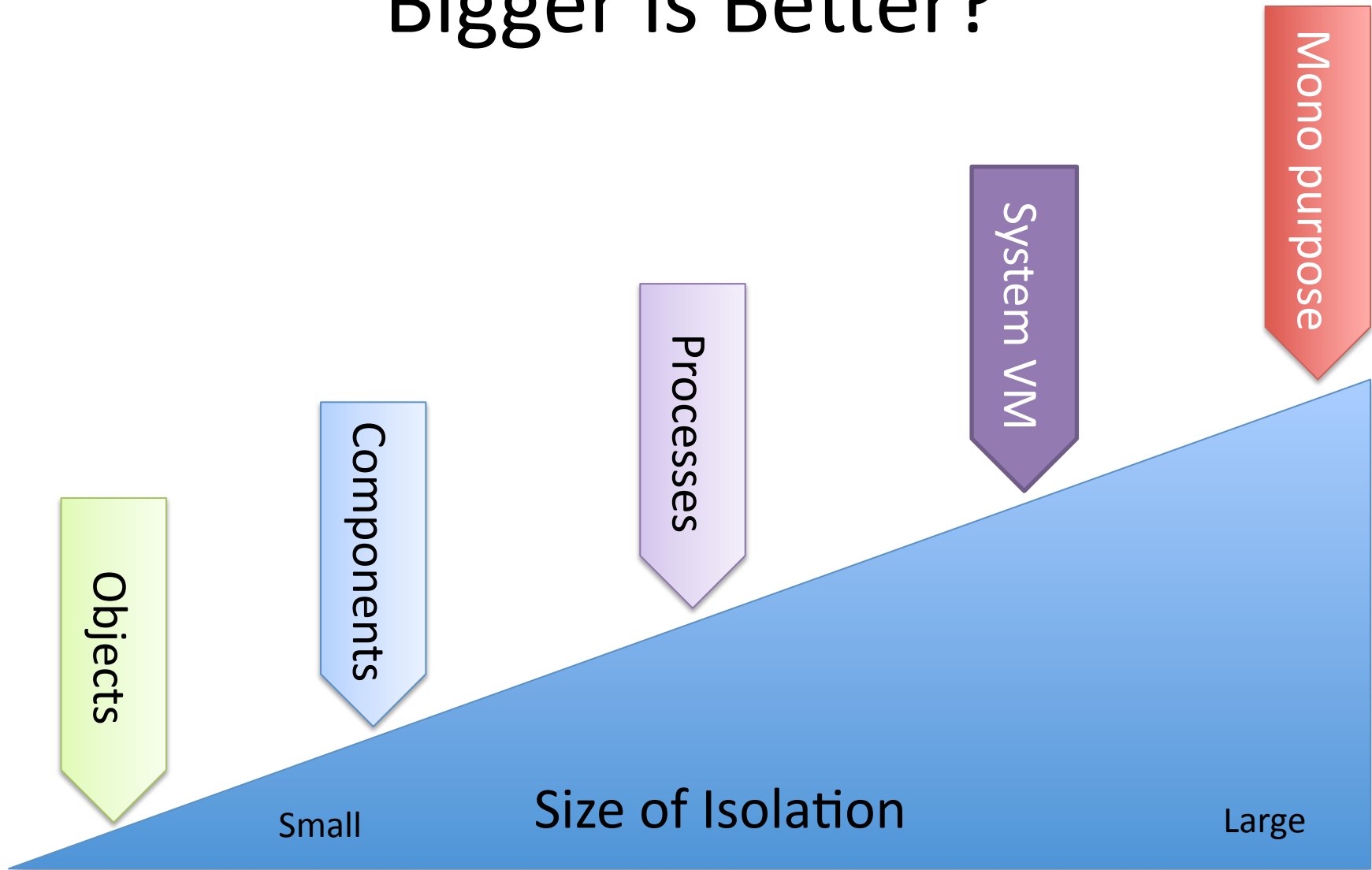


Simple Example

- Logger
- Write a message out to a log file
 - Important messages could be lost?
 - Can the disk fill, and block the application?
 - Could logger be used to write to wrong file?
 - Logger could consume memory?
 - Who can see the logs?



Bigger is Better?



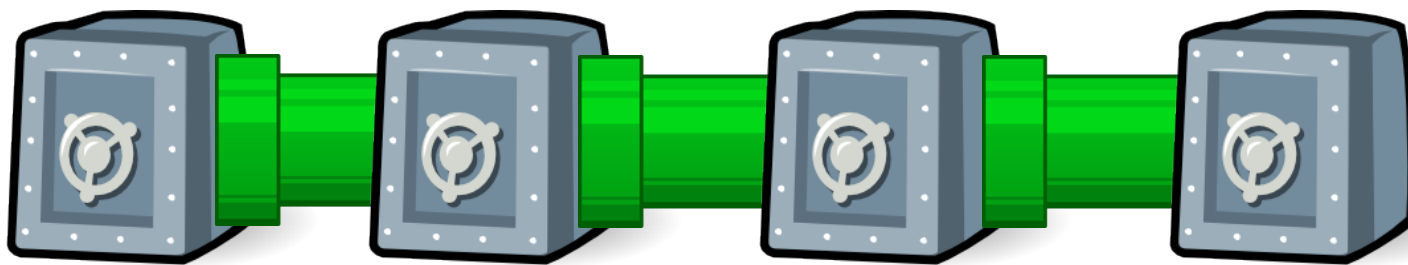
Virtual Components

- Push large protection mechanism, to smaller abstraction
- Make interaction explicit
- Virtual Component:
 - Component size virtual machine
 - VMs at the programming language and thread level
- Guarded with specific permissions
 - access to explicit typed interfaces is enforced by the system
- Virtual Components communicate events asynchronously
- The runtime enforces memory isolation between virtual components
- No shared state



Inter-Component Programming Model

- How do you *really* build systems this way?
- Who talks to who
- How do we compose components?
- How do we abstract complexities of communication?



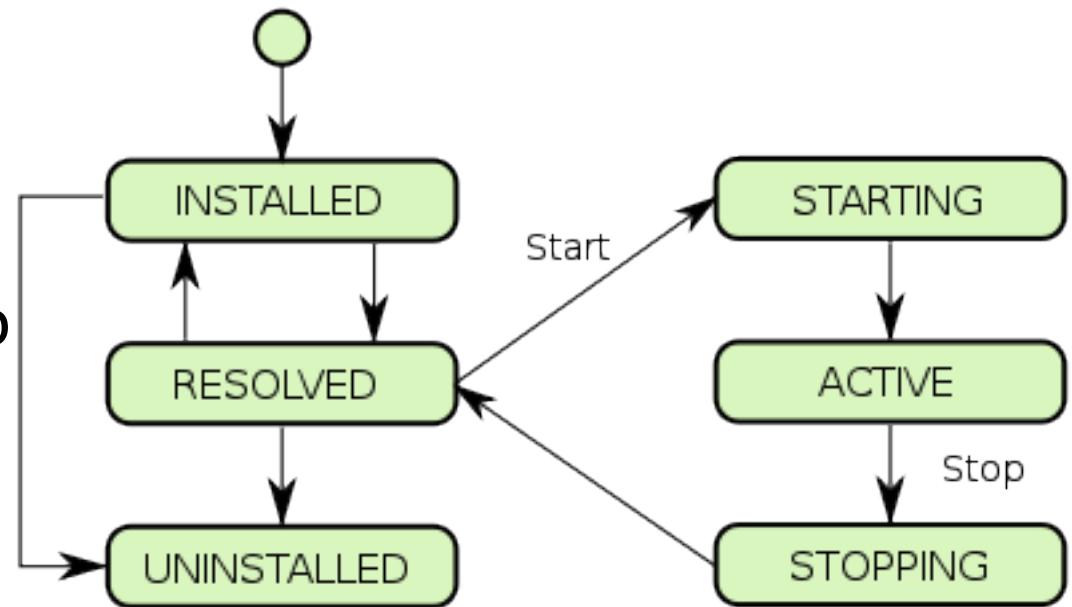
Communication

- Ultimate scalability of the system is impacted by communication mechanism
- Component level parallelism
- Problem Spots:
 - TLB and cache coherency overhead, cache misses
 - Latency
 - Performance isolation
 - Reliability
- Asynchronous communication primitive



Composition?

- Avoid hardwired dependency graphs
- OSGi?
- Lifecycle events map to component runtime states [Rellermeyer07]
- POLA



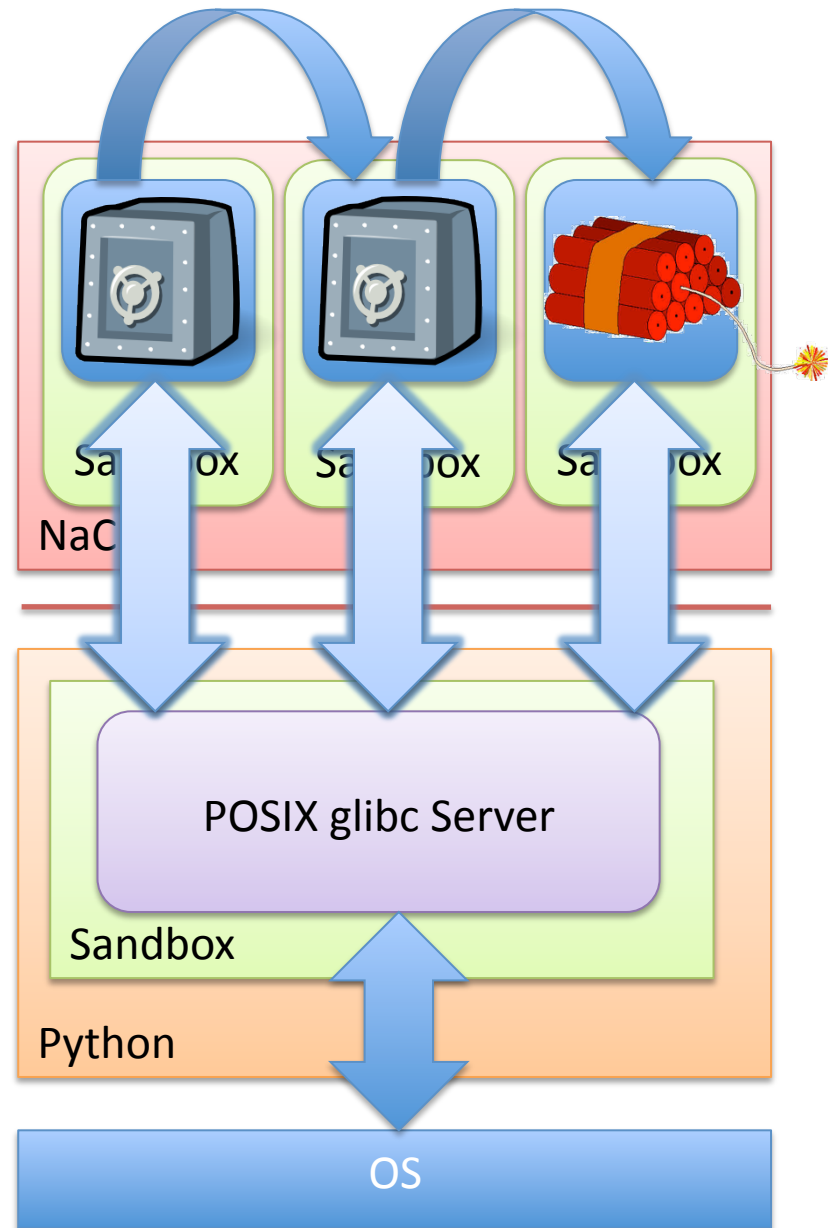


Software Fault Isolation (NaCl)

- Run real x86 (x86_64, ARM), but verify it first
 - You can't verify x86, so restrict it a little so you can
 - Modified compiler, client to verify code at load time
- All interactions are guaranteed to go through the a trampoline interface
- Google implemented this: Native Client
- Two layer sandbox
- Real X86 with assembly, SSE, threading etc

Lind

- A SFI isolated component model
- Components are executable binary code
- Interaction with underlying OS and other components is strictly controlled
- Service runtime written in Python sandbox, provides POSIX API
- Trusted Computing Base!

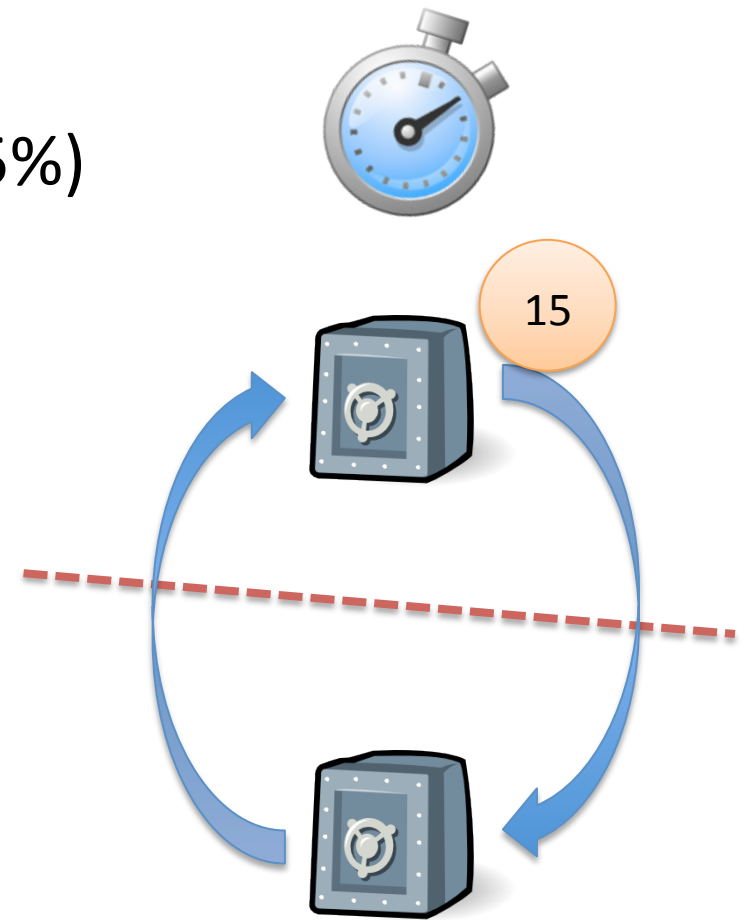


Evaluation Techniques

- With so many tradeoffs how do we assess the value of a virtual component model?
- How to compare with other work?
- Tradeoff Space:
 - Performance
 - Isolation
 - Security
 - Composition

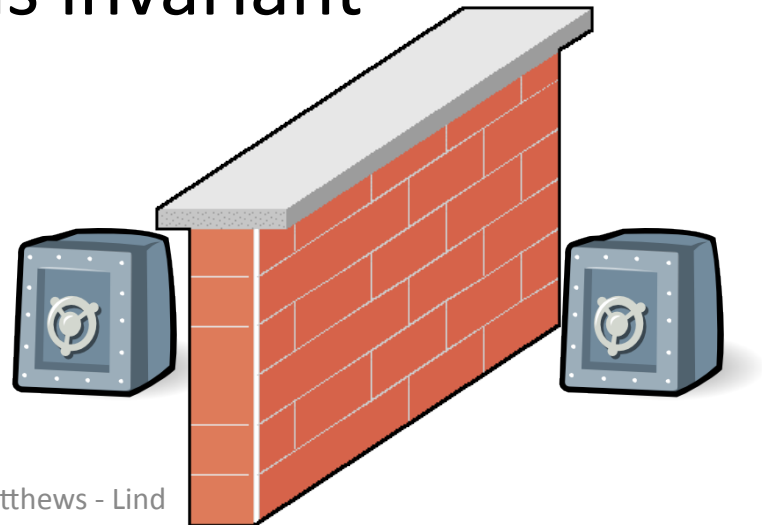
Performance

- Characterising the costs:
 - Execution overhead (NaCl \rightarrow 5%)
 - Communication overhead
 - Micro benchmarks
 - Integer ping pong



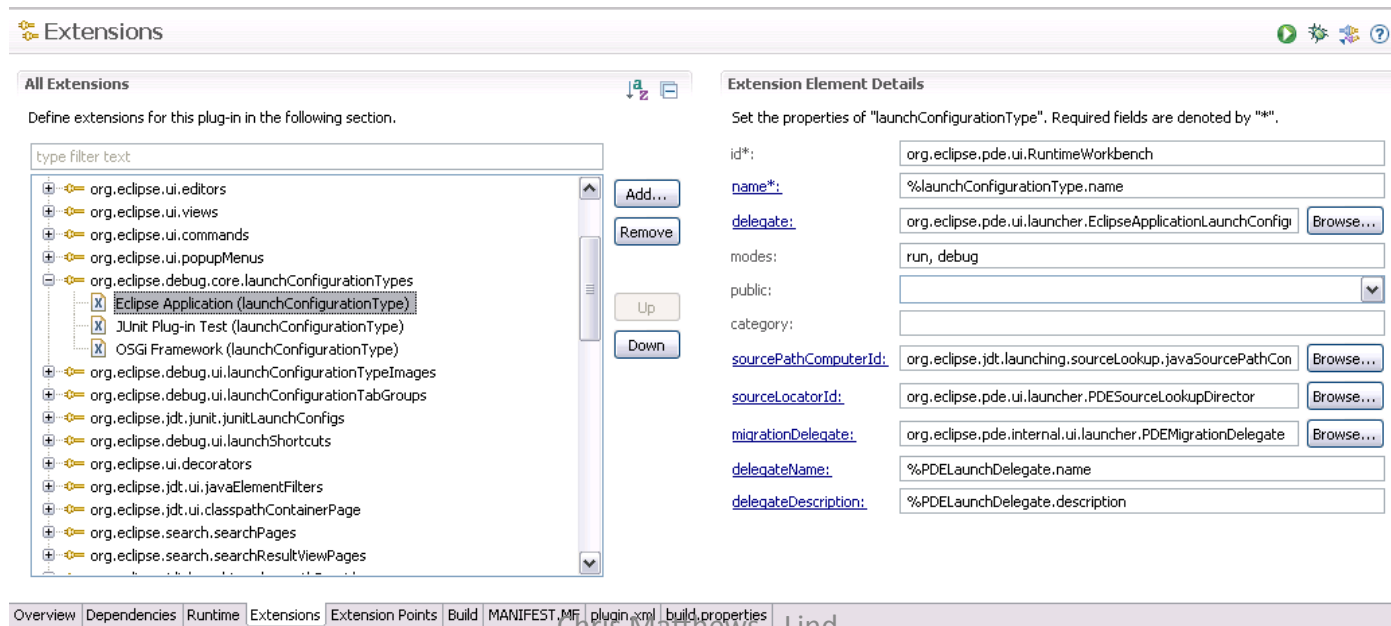
Isolation

- Hard to measure spatial isolation
 - How good is the underlying isolation
- Micro benchmarks
 - temporal isolation
- Try to show isolation is invariant











Composition

- Composition as a first class citizen?
- Can modern tools and practices be applied in the same way as with current component models?



Security

- POLA - how well do we achieve it?
- Which current attacks do we retard or stop

Attack	Outcome
Resource Exhaustion	
Buffer Overflow	
CSRF	
Injection Attacks	
Privilege Escalation	
Information Leakage	
Side Channel Attacks	
TOCTTOU	

Evaluation Summary

Trait	Metric
Performance	Micro benchmarks
Isolation	Benchmarks and invariance
Security	What does it fix?
Composition	Current techniques apply?

Questions

- Would you use it?
 - Even if it was really slow? (100x slower)
- What existing composition mechanisms might apply here?
- How do we deal with the coupling between structure, performance, isolation and security?
- Applications where this would work?

