



# Systems and Internet Infrastructure Security

Network and Security Research Center  
Department of Computer Science and Engineering  
Pennsylvania State University, University Park PA

## Justifying Integrity Using a Virtual Machine Verifier

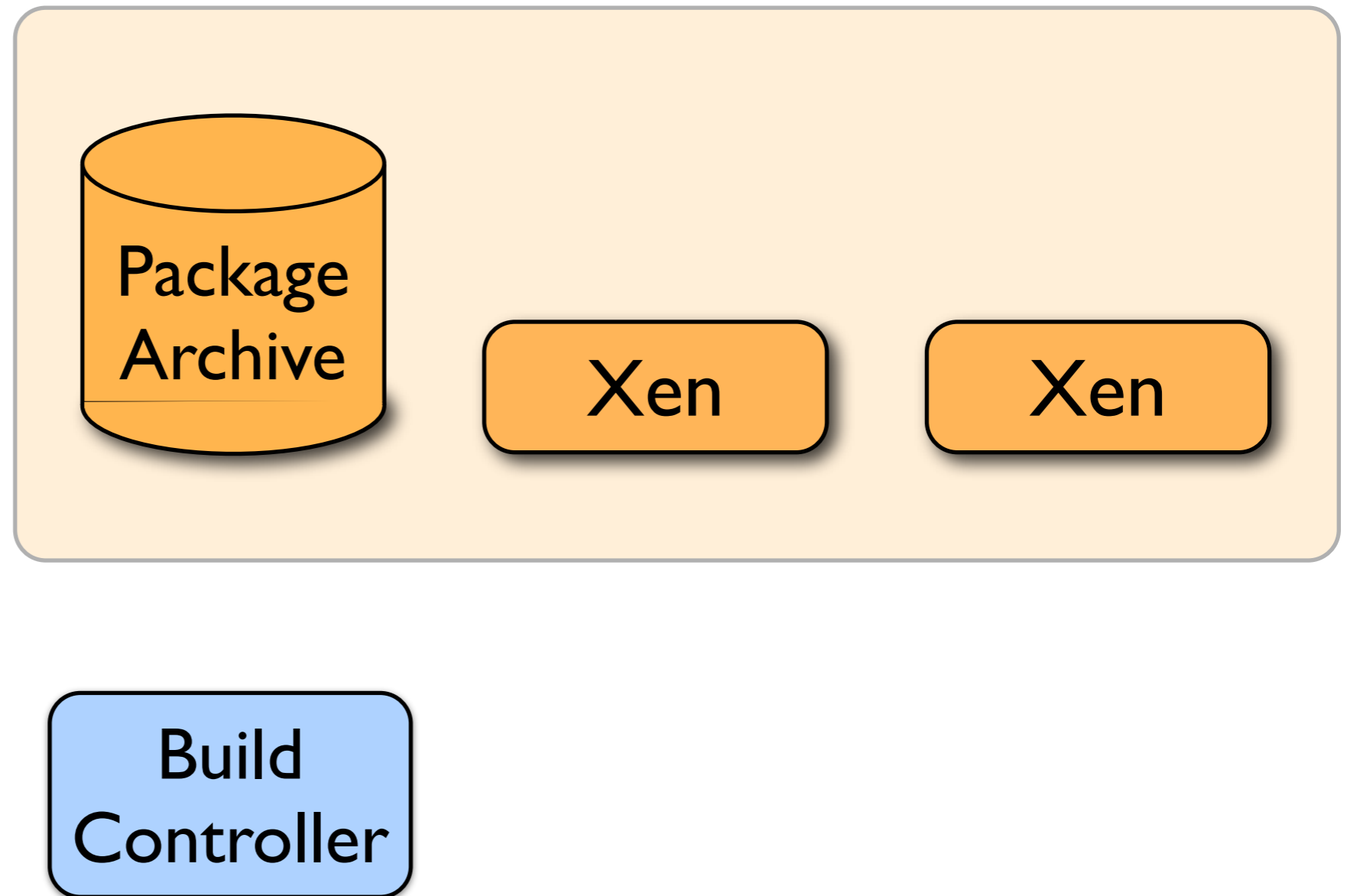
**Joshua Schiffman**, Thomas Moyer, Christopher Shal,  
Trent Jaeger, and Patrick McDaniel  
ACSAC '09

# Cloudy Horizons

- Utility-based cloud computing is attracting small businesses and developers
- Clouds offer an on-demand virtualization infrastructure to run distributed applications
- How can we verify that application components are behaving as expected?



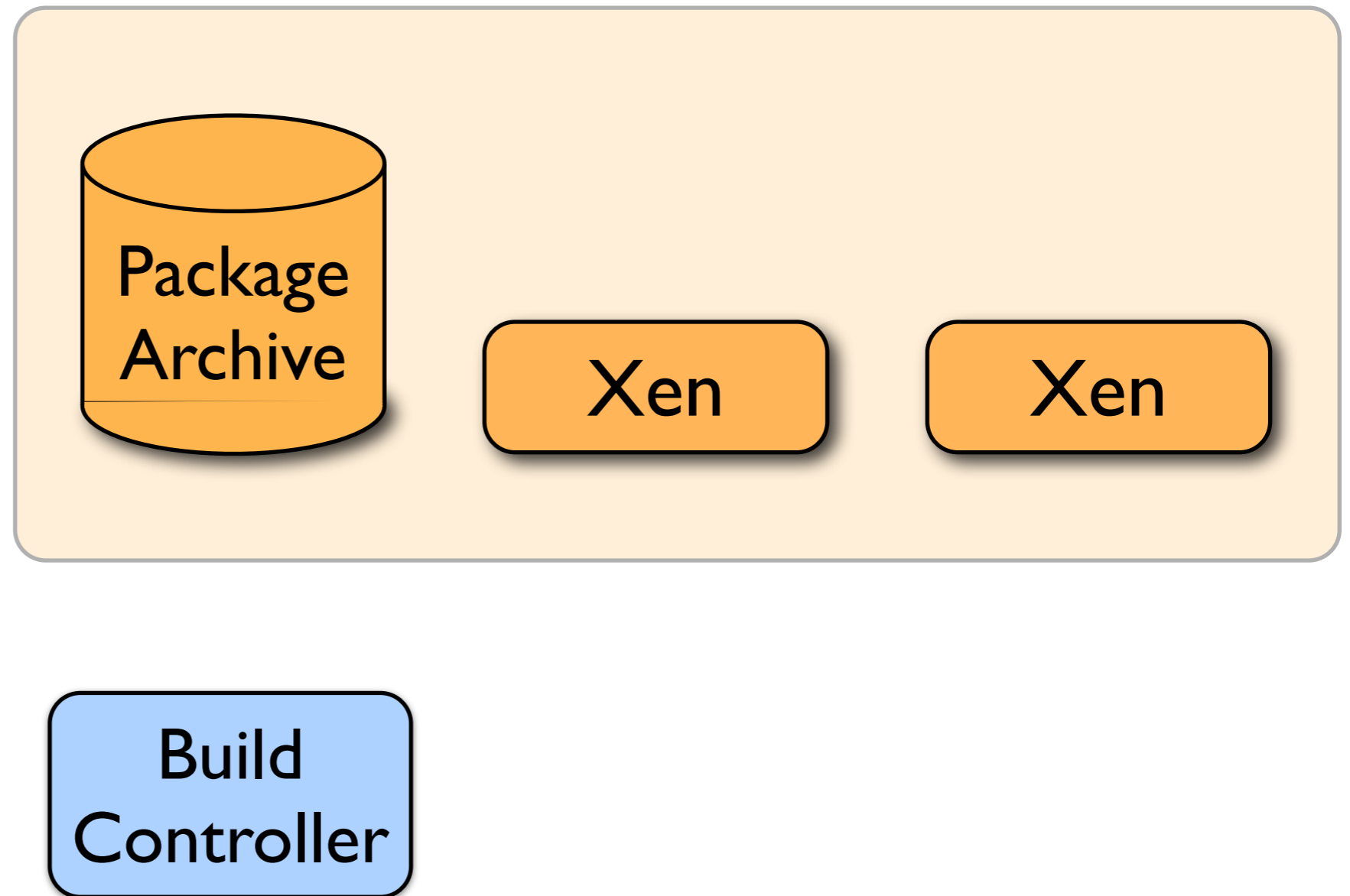
## Cloud Infrastructure



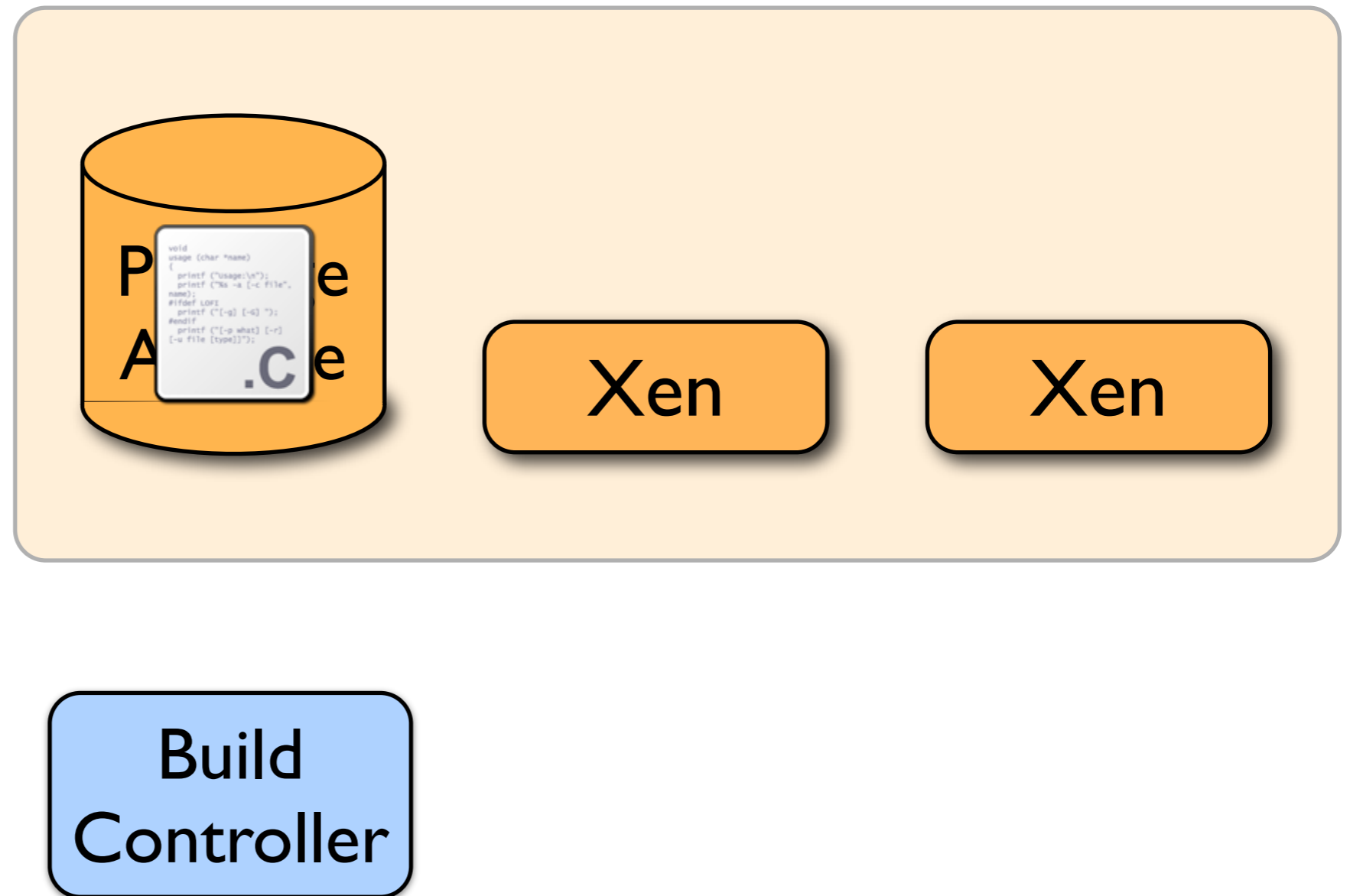
## Cloud Infrastructure

```
void  
usage (char *name)  
{  
    printf ("usage:\n");  
    printf ("%s -a [-c file",  
    name);  
    #ifdef LDF  
    printf ("[-g] [-d ]");  
    #endif  
    printf ("[-p what] [-r]  
    [-u file [type]]");  
}
```

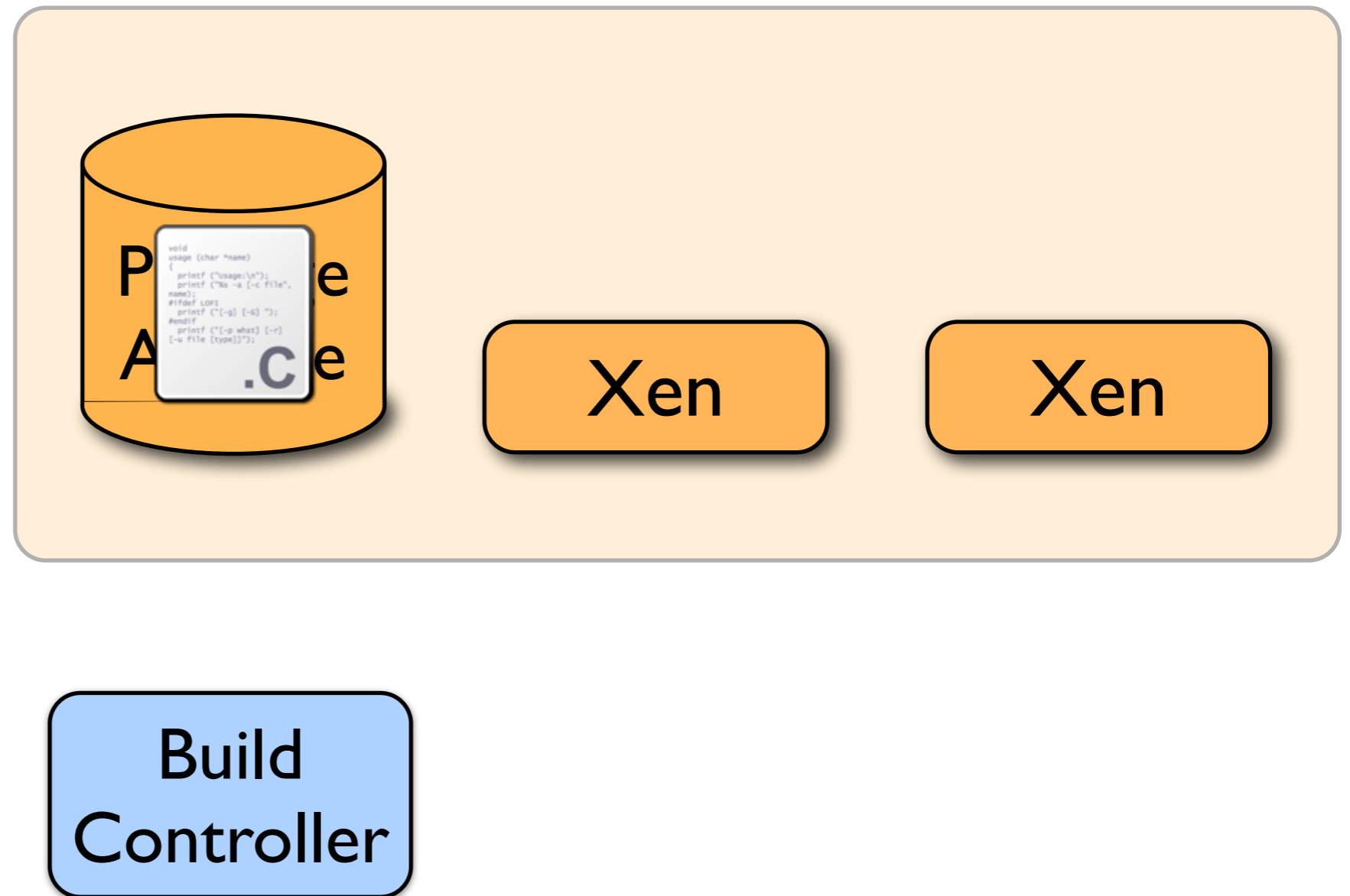
.C



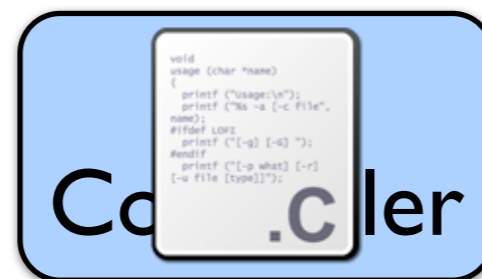
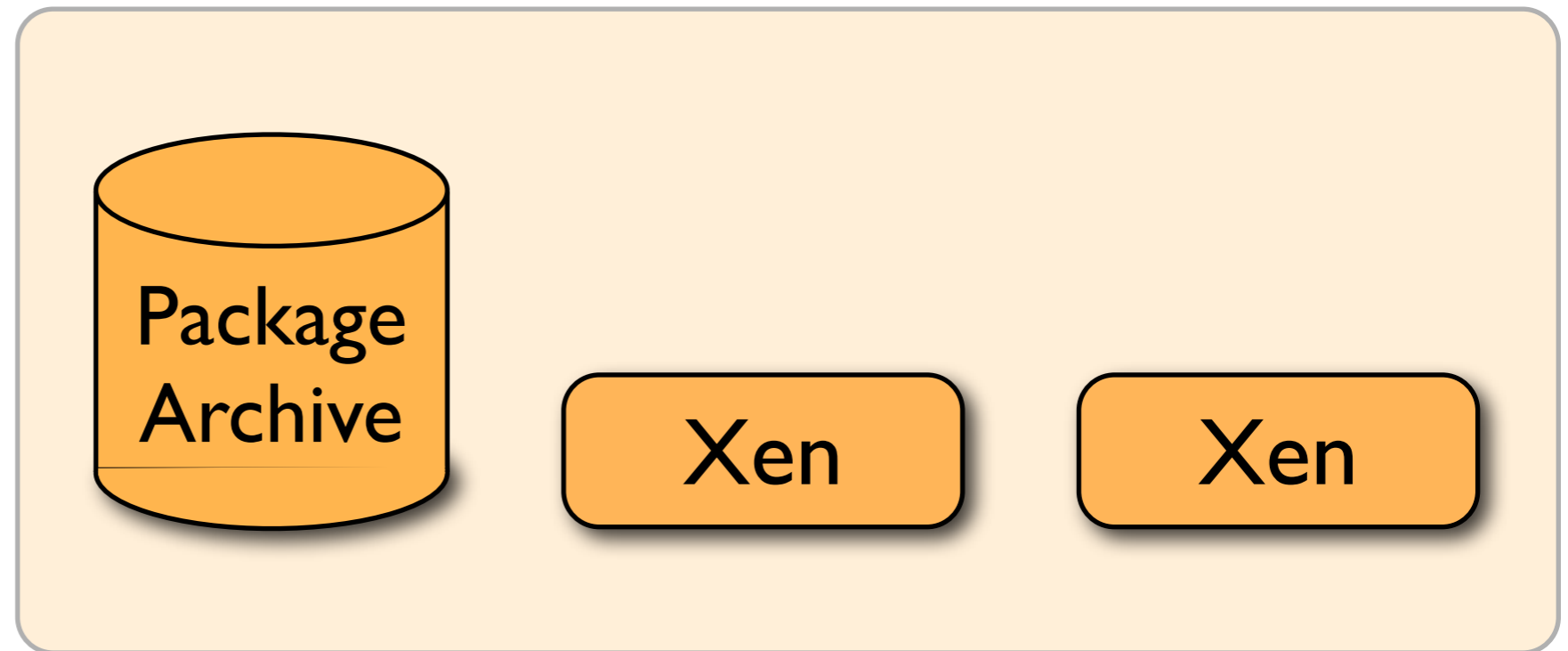
## Cloud Infrastructure



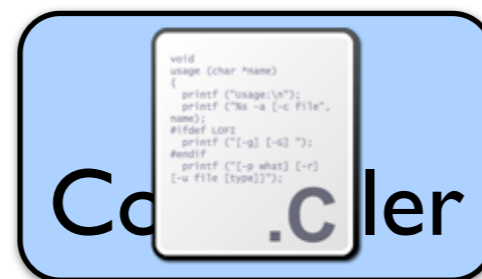
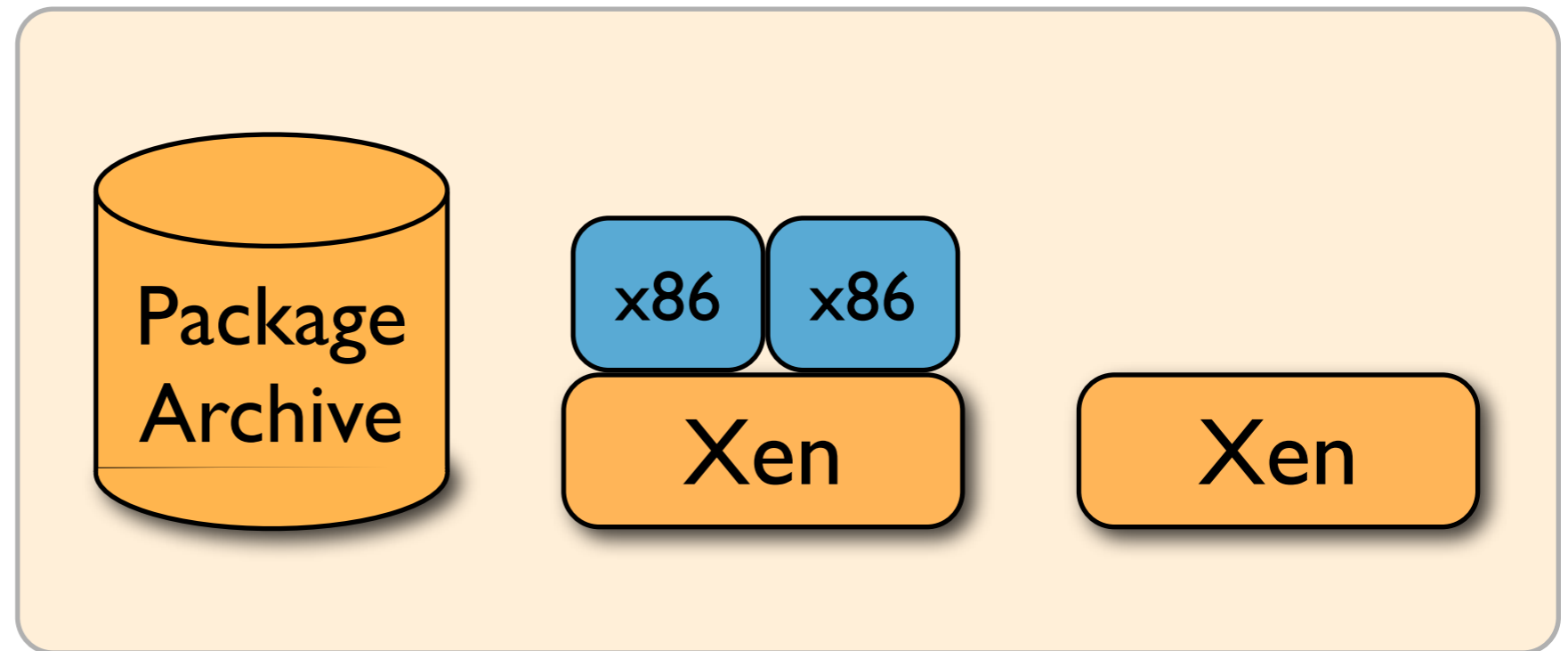
## Cloud Infrastructure



## Cloud Infrastructure

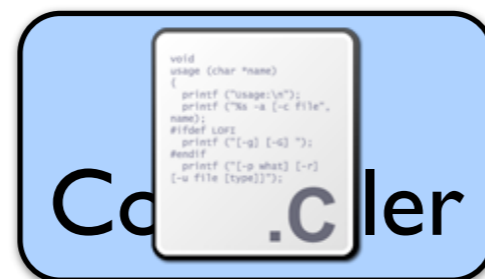
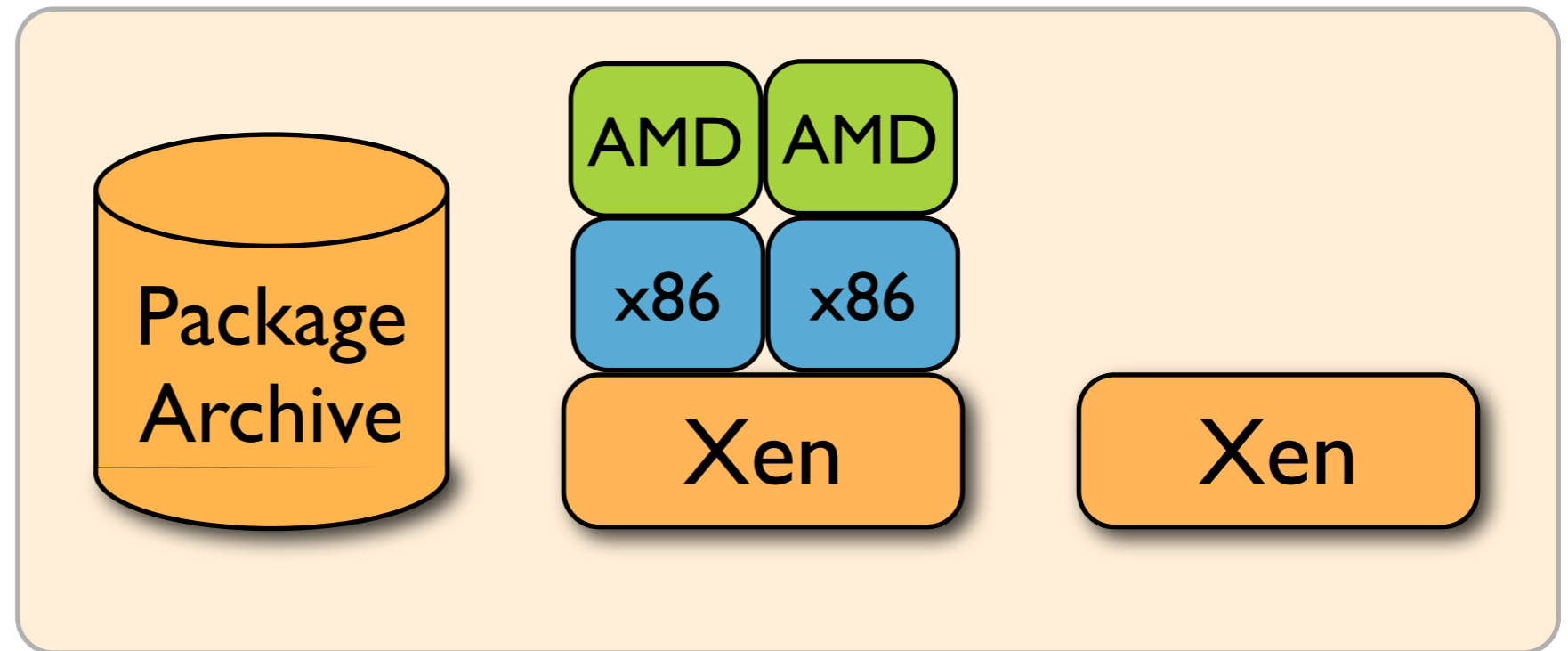


## Cloud Infrastructure



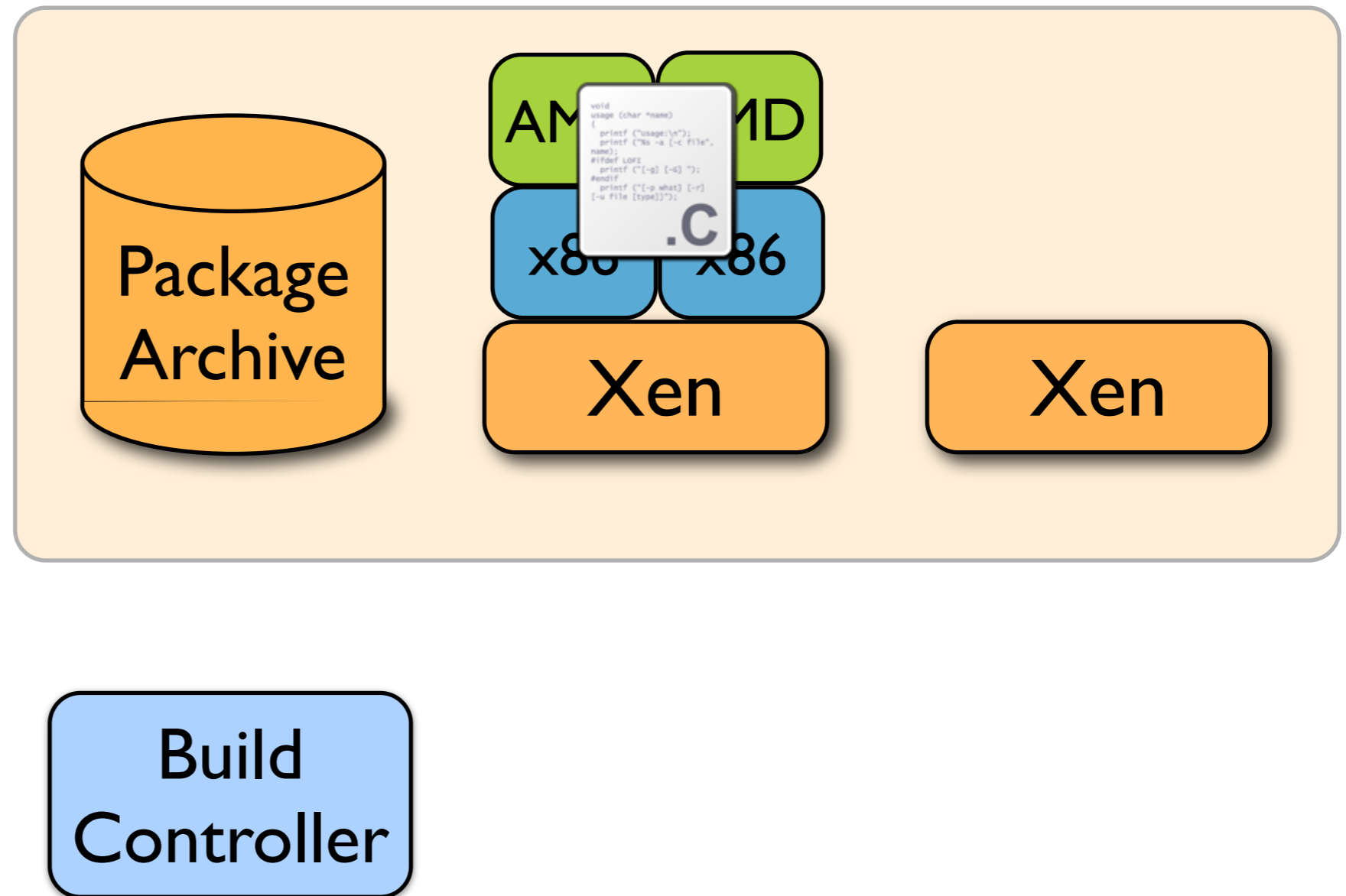
# Distributed Compilation

## Cloud Infrastructure



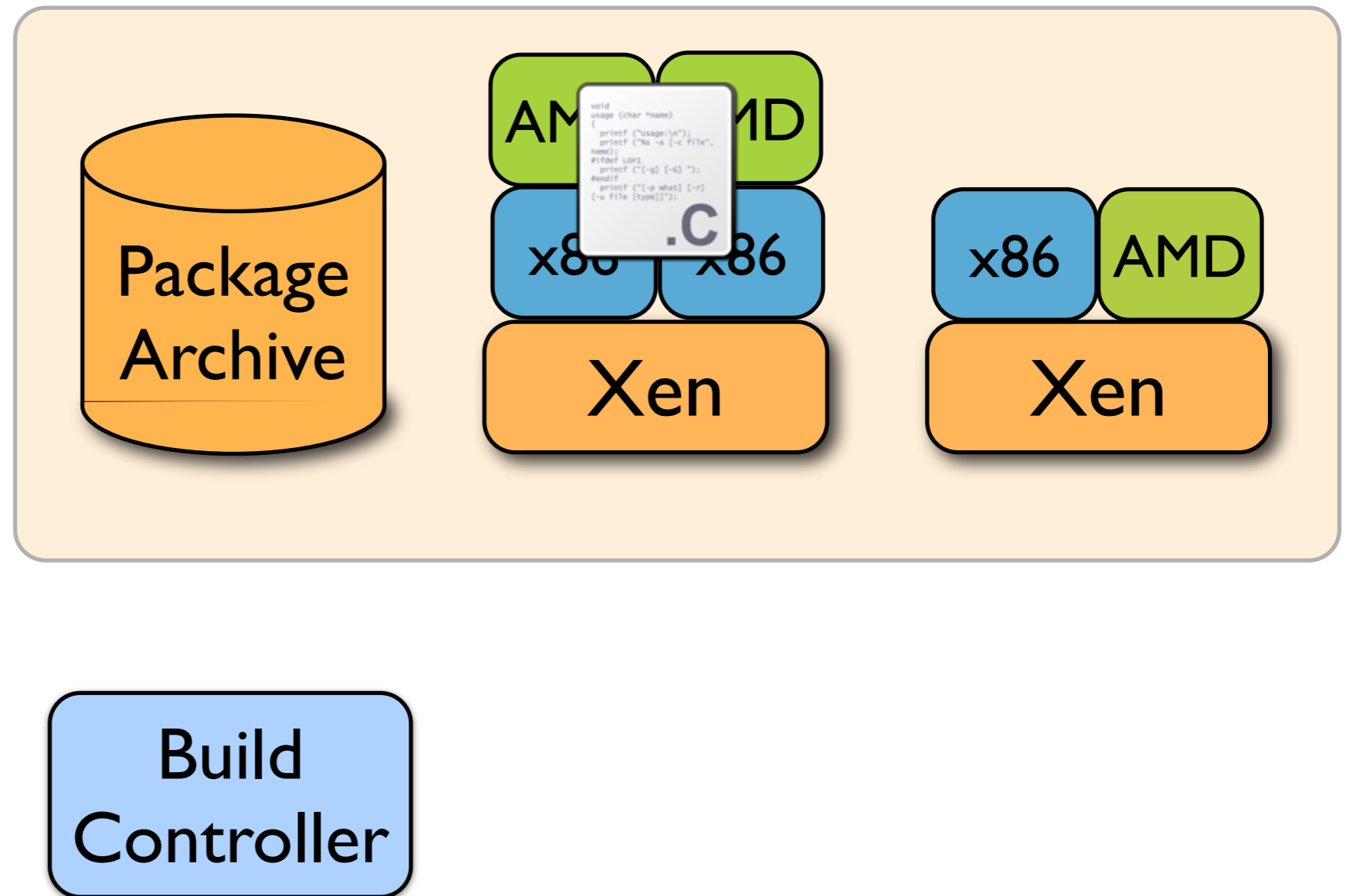
# Distributed Compilation

## Cloud Infrastructure



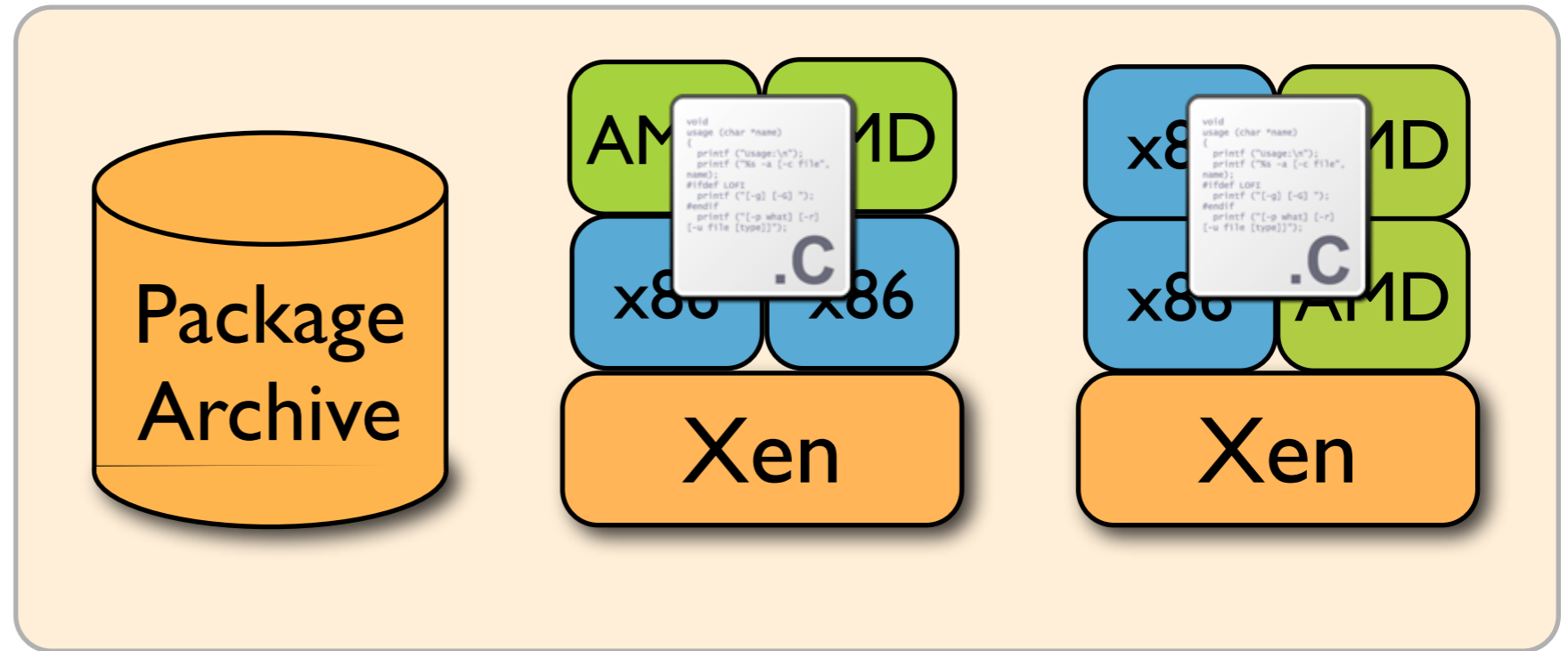
# Distributed Compilation

## Cloud Infrastructure



# Distributed Compilation

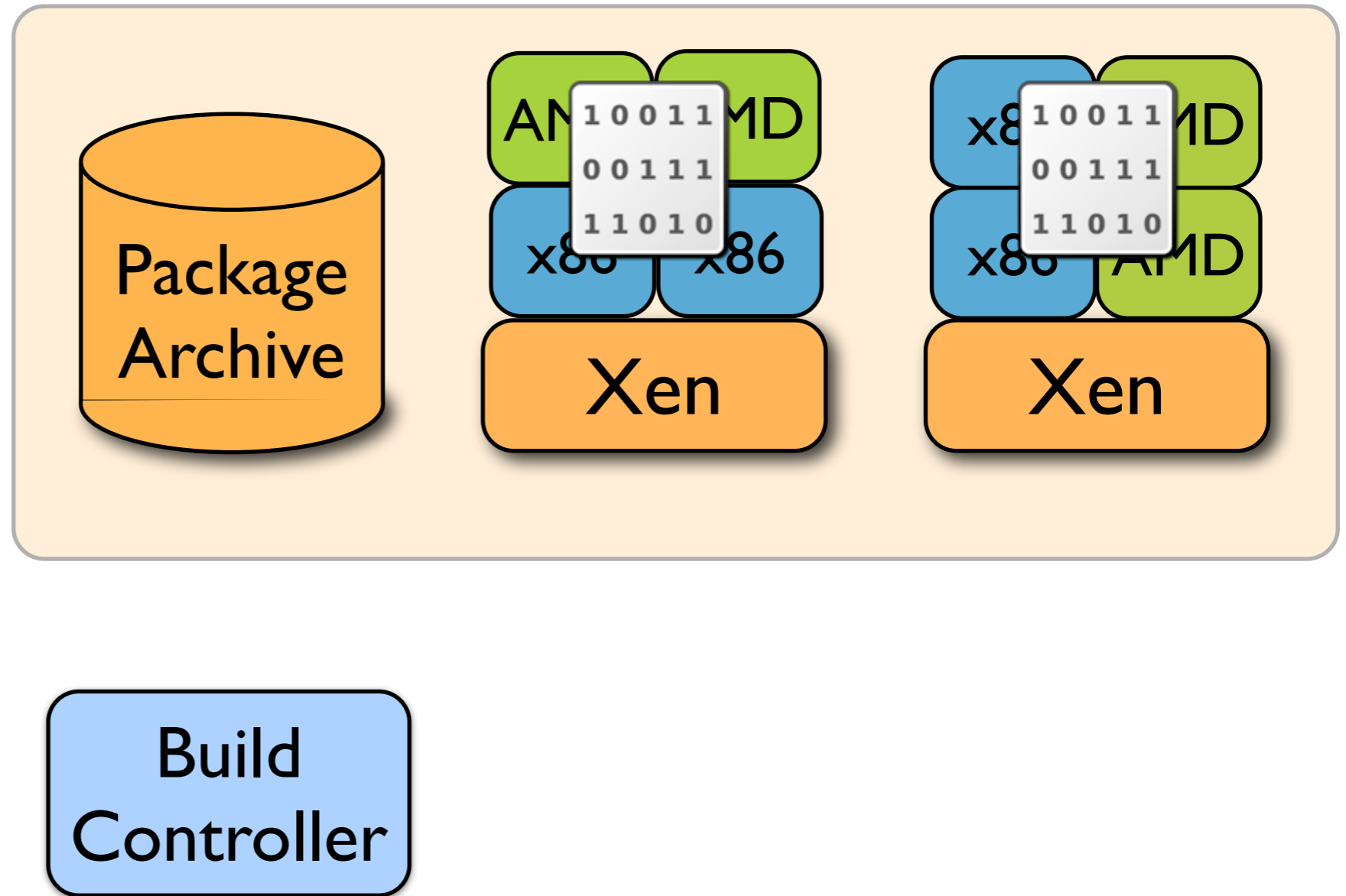
## Cloud Infrastructure



Build  
Controller

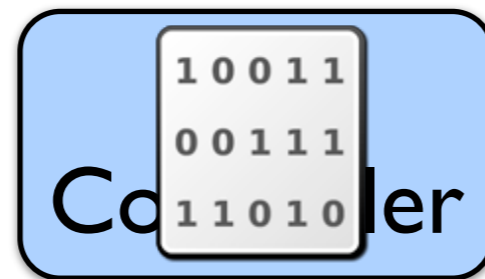
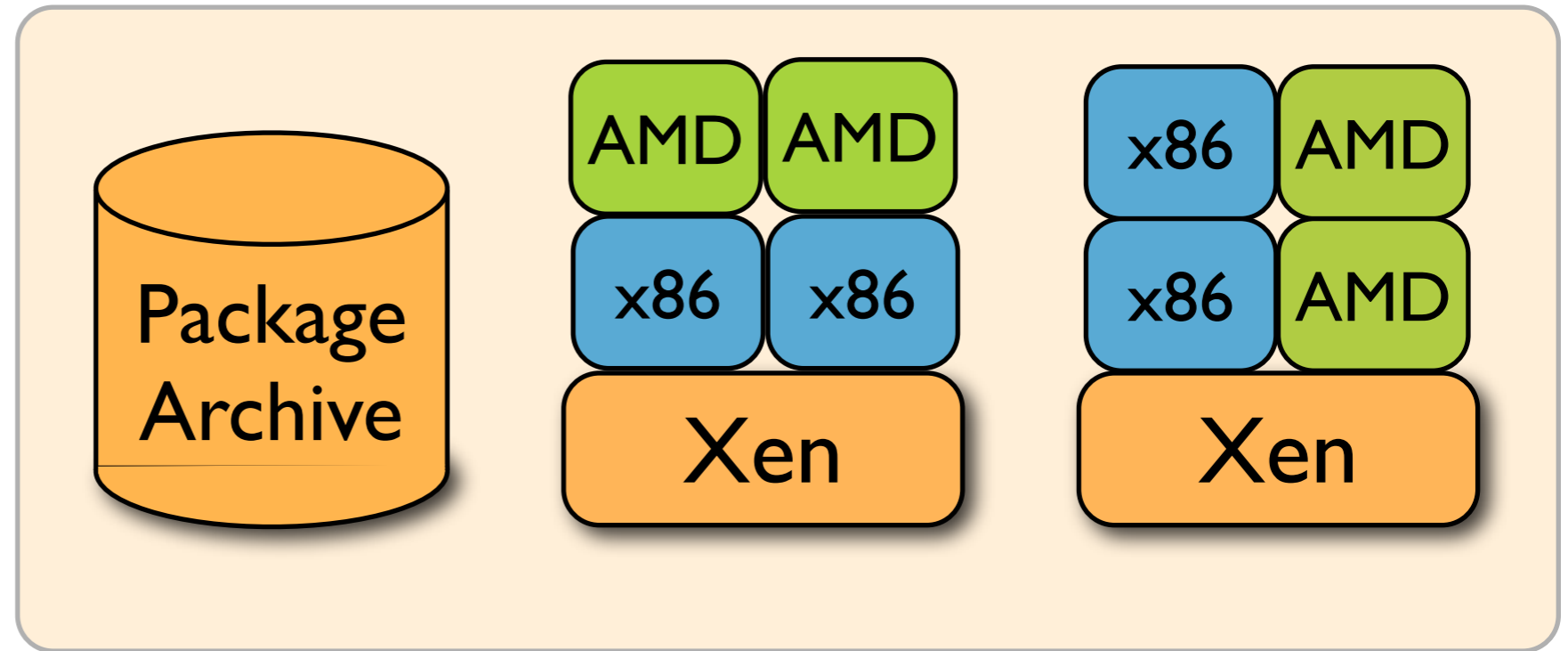
# Distributed Compilation

## Cloud Infrastructure



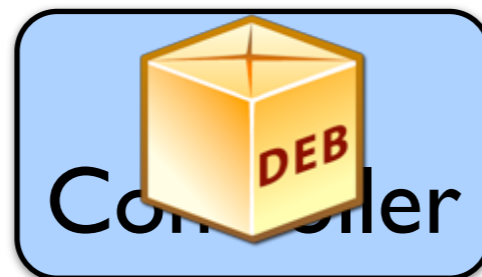
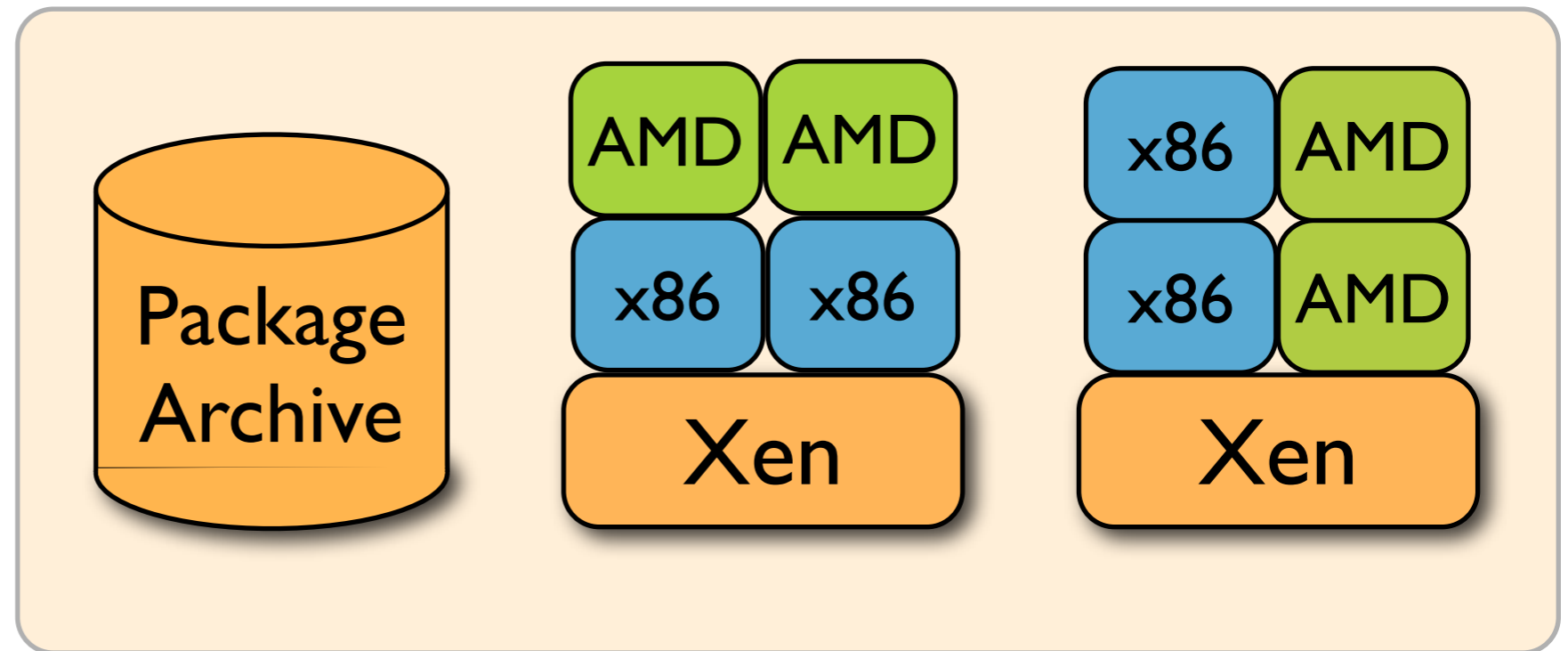
# Distributed Compilation

## Cloud Infrastructure



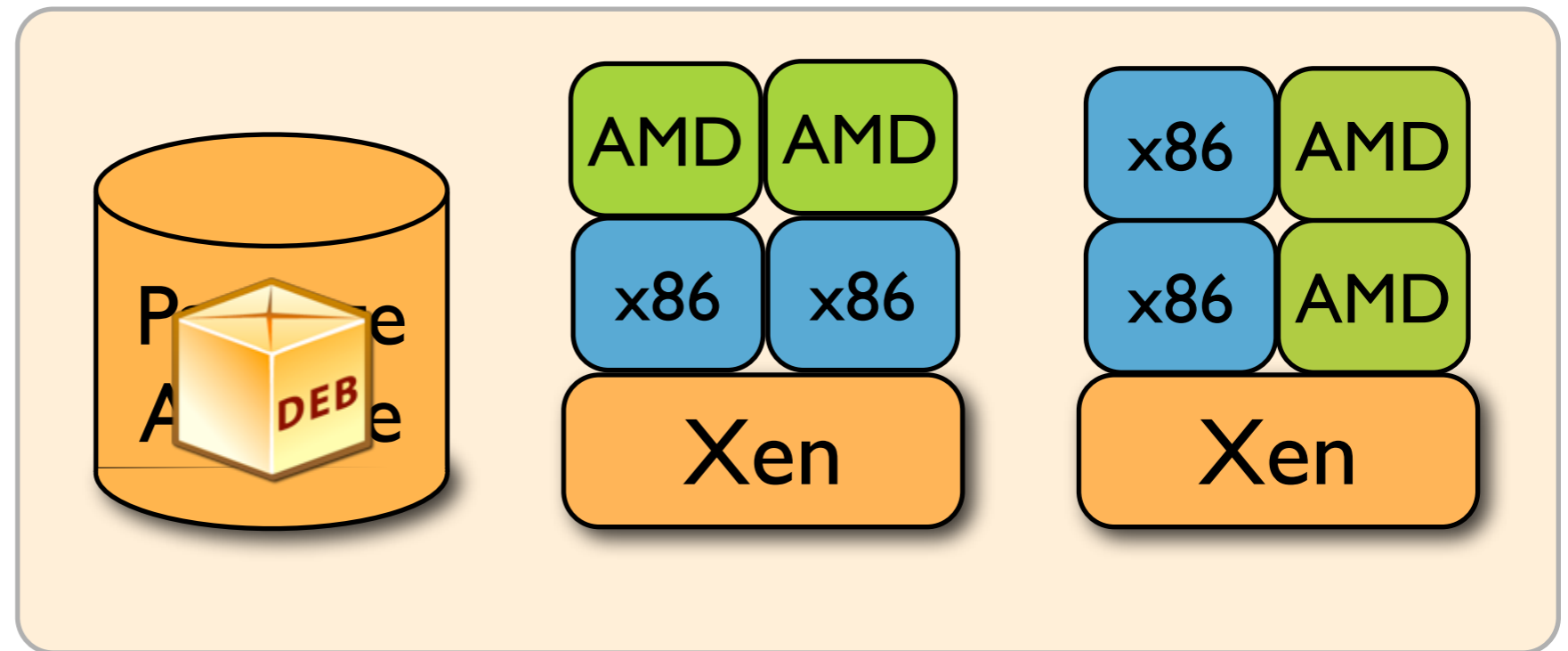
# Distributed Compilation

## Cloud Infrastructure



# Distributed Compilation

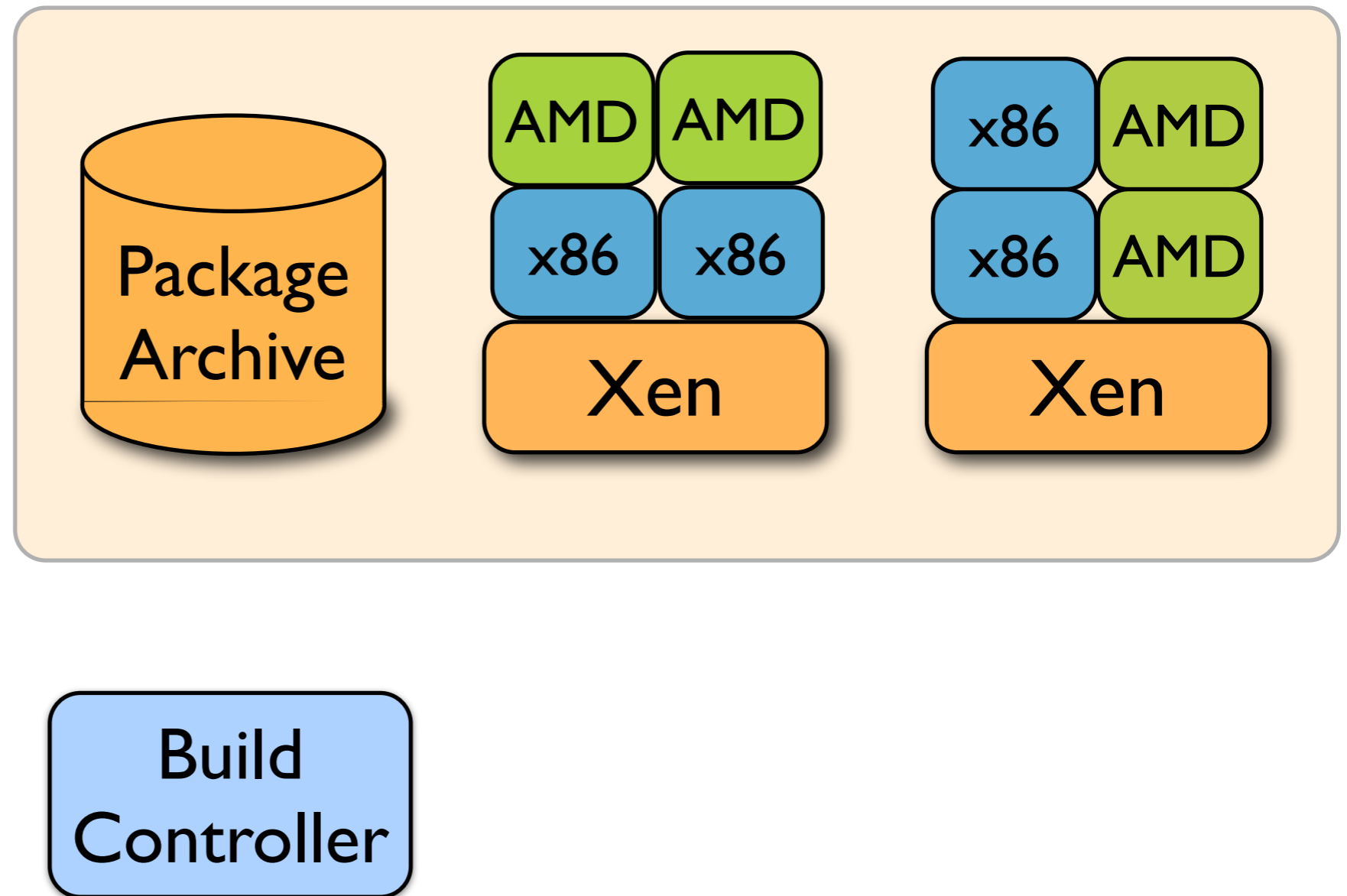
## Cloud Infrastructure



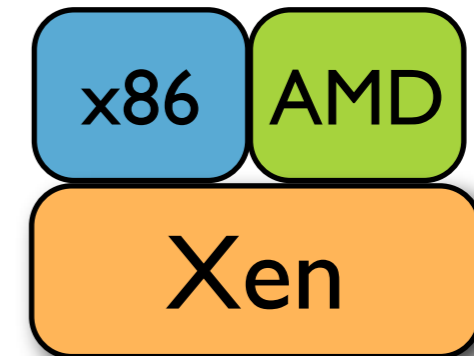
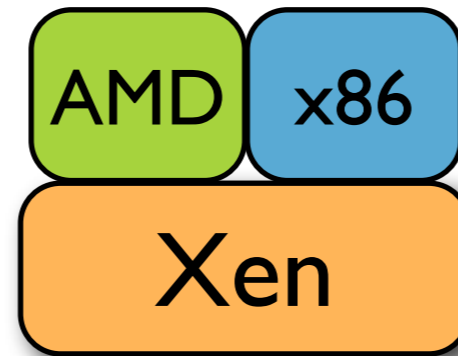
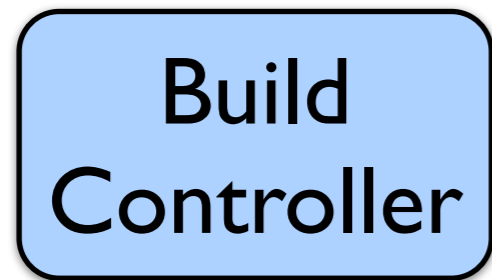
Build  
Controller

# Distributed Compilation

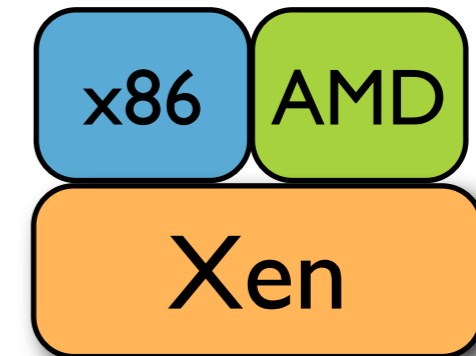
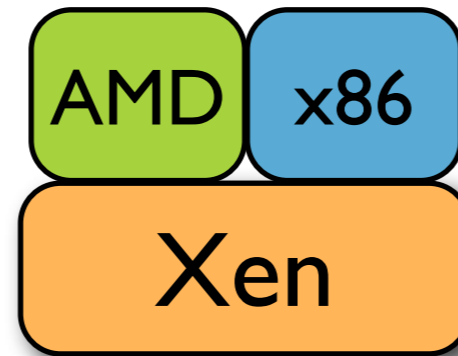
## Cloud Infrastructure



# Threat Model

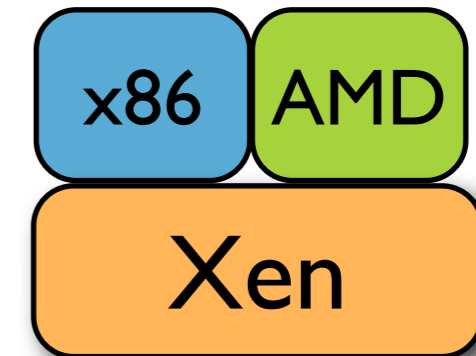
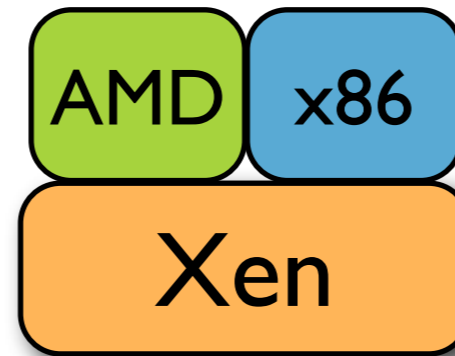
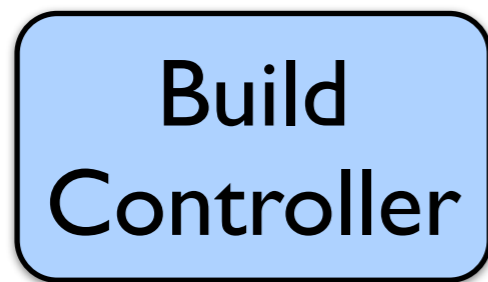


# Threat Model



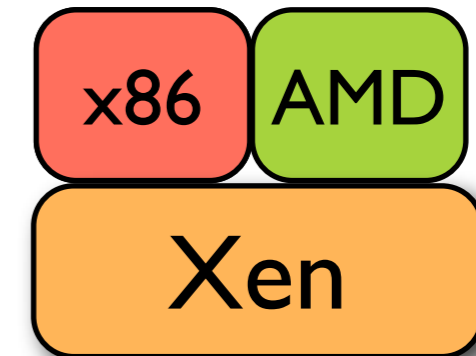
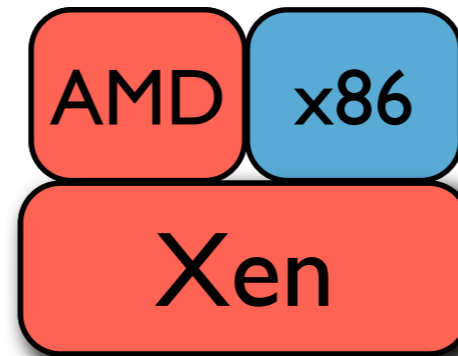
- Build controller depends on input from other VMs

# Threat Model



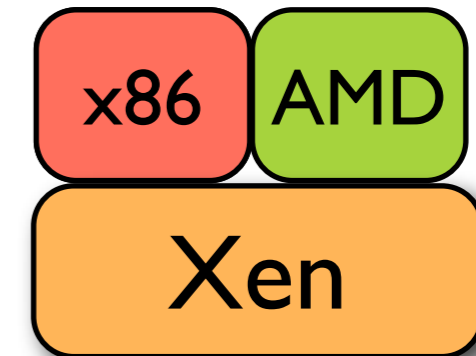
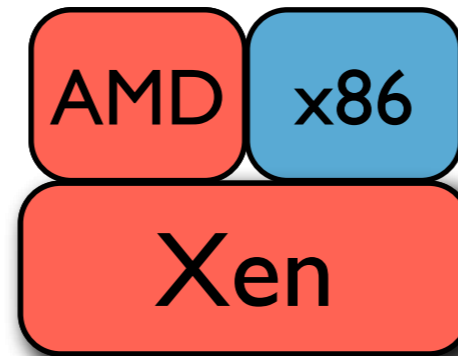
- Build controller depends on input from other VMs
  - ▶ Cloud VMs / hosts can be misconfigured or compromised

# Threat Model



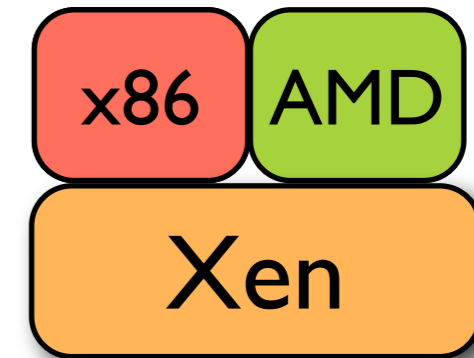
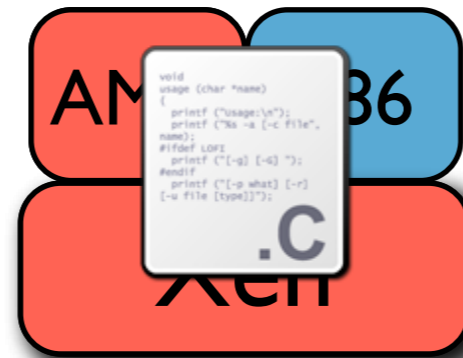
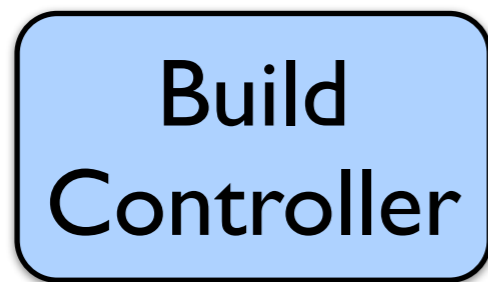
- Build controller depends on input from other VMs
  - ▶ Cloud VMs / hosts can be misconfigured or compromised

# Threat Model



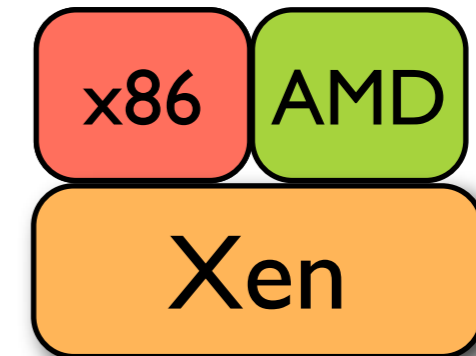
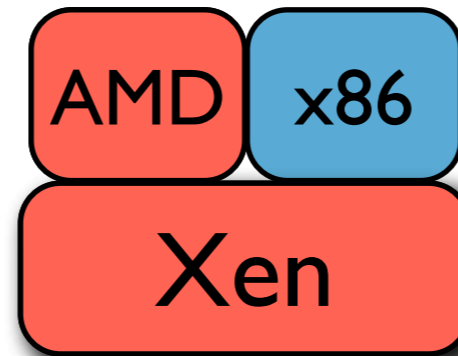
- Build controller depends on input from other VMs
  - ▶ Cloud VMs / hosts can be misconfigured or compromised
  - ▶ May generate malicious results

# Threat Model

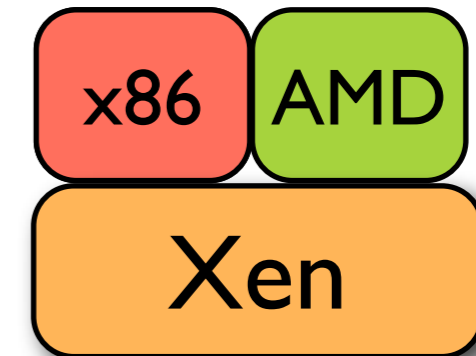
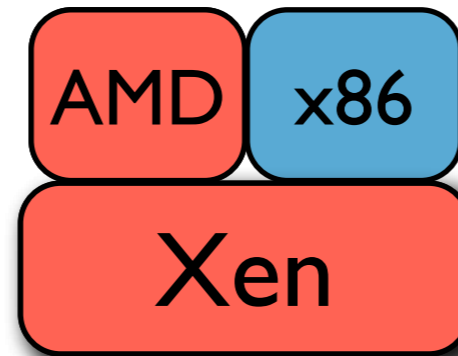
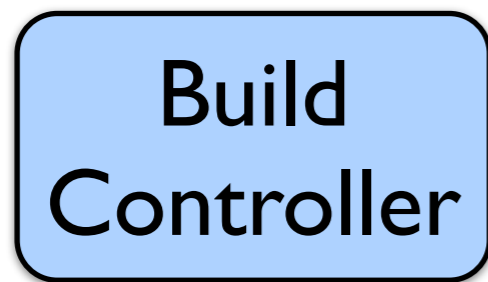


- Build controller depends on input from other VMs
  - ▶ Cloud VMs / hosts can be misconfigured or compromised
  - ▶ May generate malicious results

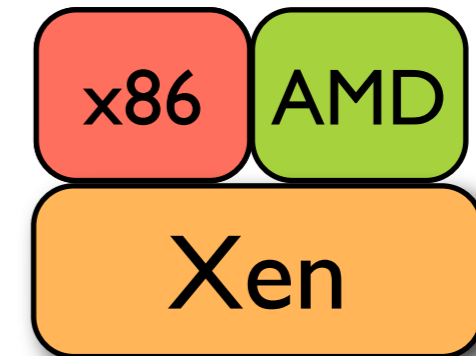
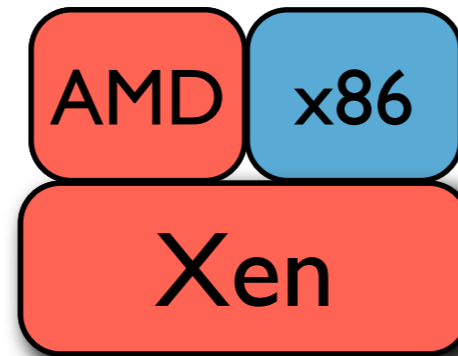
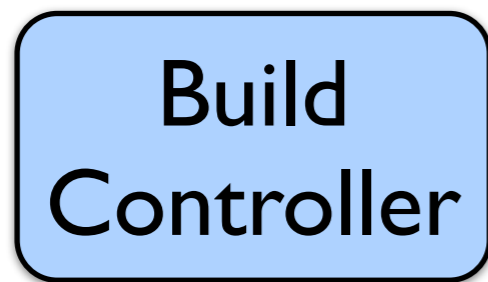
# Threat Model



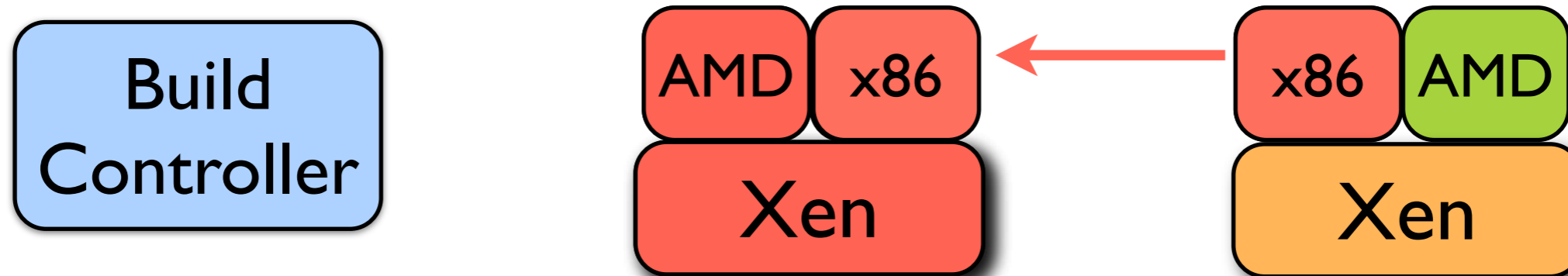
- Build controller depends on input from other VMs
  - ▶ Cloud VMs / hosts can be misconfigured or compromised
  - ▶ May generate malicious results



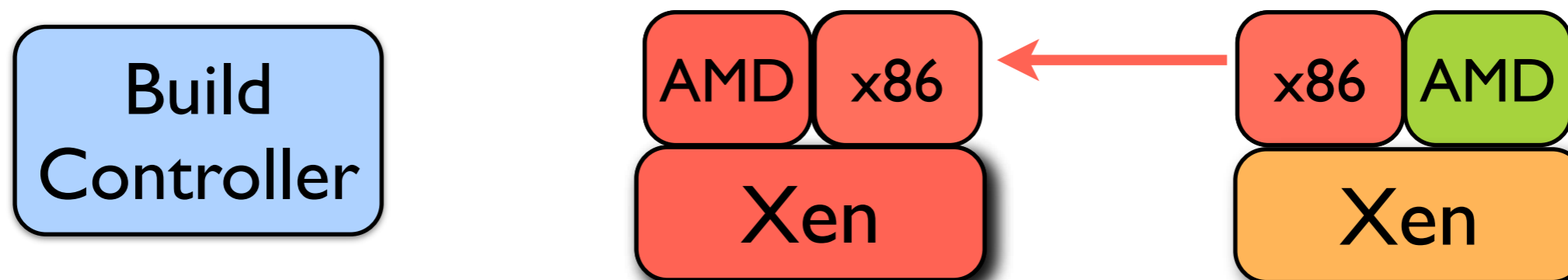
- Build controller depends on input from other VMs
  - ▶ Cloud VMs / hosts can be misconfigured or compromised
  - ▶ May generate malicious results



- Build controller depends on input from other VMs
  - ▶ Cloud VMs / hosts can be misconfigured or compromised
  - ▶ May generate malicious results
  - ▶ Other cloud instances may affect vulnerable systems



- Build controller depends on input from other VMs
  - ▶ Cloud VMs / hosts can be misconfigured or compromised
  - ▶ May generate malicious results
  - ▶ Other cloud instances may affect vulnerable systems



- Build controller depends on input from other VMs
  - ▶ Cloud VMs / hosts can be misconfigured or compromised
  - ▶ May generate malicious results
  - ▶ Other cloud instances may affect vulnerable systems
- How can we **verify** that the cloud VMs are **correct** and **provide trustworthy inputs**?

- Goal: measure against classical integrity models
  - ▶ Current approaches are incomplete
- **Clark-Wilson** aims to protect high integrity data from low integrity input and processes
  - ▶ *Unconstrained Data Items* (UDI): low integrity data
  - ▶ *Constrained Data Items* (CDI): high integrity data
    - Verified by Integrity Verification Procedures (IVPs)
  - ▶ *Transformation Procedures* (TP) modify CDIs
    - TPs must be *formally assured* to behave correctly
    - All interfaces discard or upgrade UDIs

- Practical integrity models are less heavyweight
  - ▶ CW-Lite [Shankar '06], UMIP [Li '07], PPI [Sun '08]
- We focus on verifying **CW-Lite integrity**
  - ▶ Requires *identification of trusted code* in TPs
  - ▶ Only interfaces that receive UDIs are filtered
- Build controller must verify for each VM
  - ▶ CDIs pass an IVP
  - ▶ TP are trustworthy
  - ▶ UDIs only enter through filtering interfaces

- Measure integrity-relevant operations sufficient to prove CW-Lite for Build VMs
  - ▶ BIND: measure integrity data based on generating code
  - ▶ Satem: enforce execution of specified code
  - ▶ PRIMA: ensure that high integrity data is modified by only trusted processes
- Enable attestation of integrity to a remote party

- Measure integrity-relevant operations necessary to prove CW-Lite for Build VMs
  - ▶ BIND: measure integrity data based on generating code
    - IVP for verifying CDIs' integrity
  - ▶ Satem: enforce execution of specified code
    - Ensure use of only certified TPs
  - ▶ PRIMA: ensure that high integrity data is modified by only trusted processes
    - Ensure that only TPs modify CDIs and handle UDIs securely
- Each are incomplete and still missing some function necessary to build CW-Lite proofs

# Our Solution

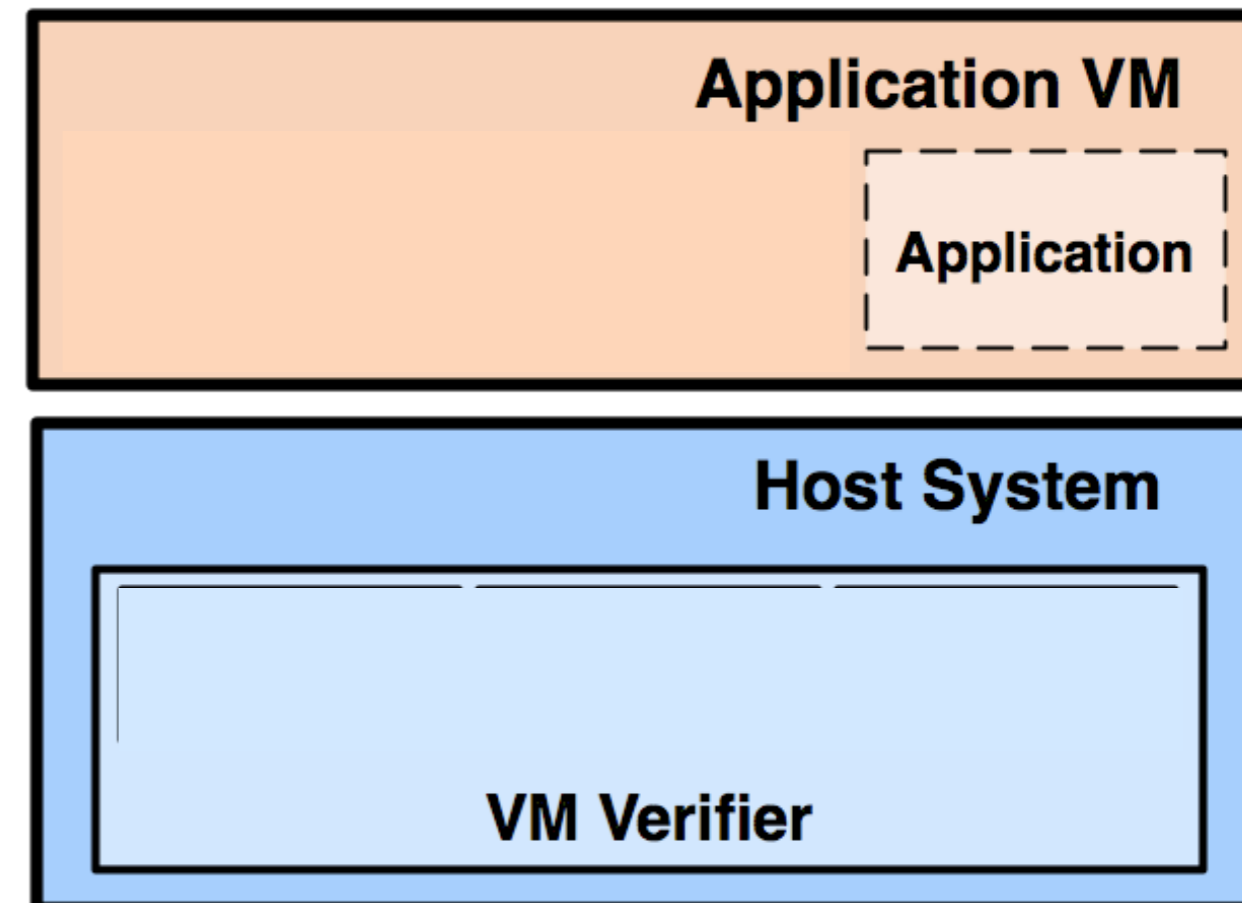


# Our Solution

- We **enforce** and enable **verification** of CW-Lite on a VM system for a particular application policy

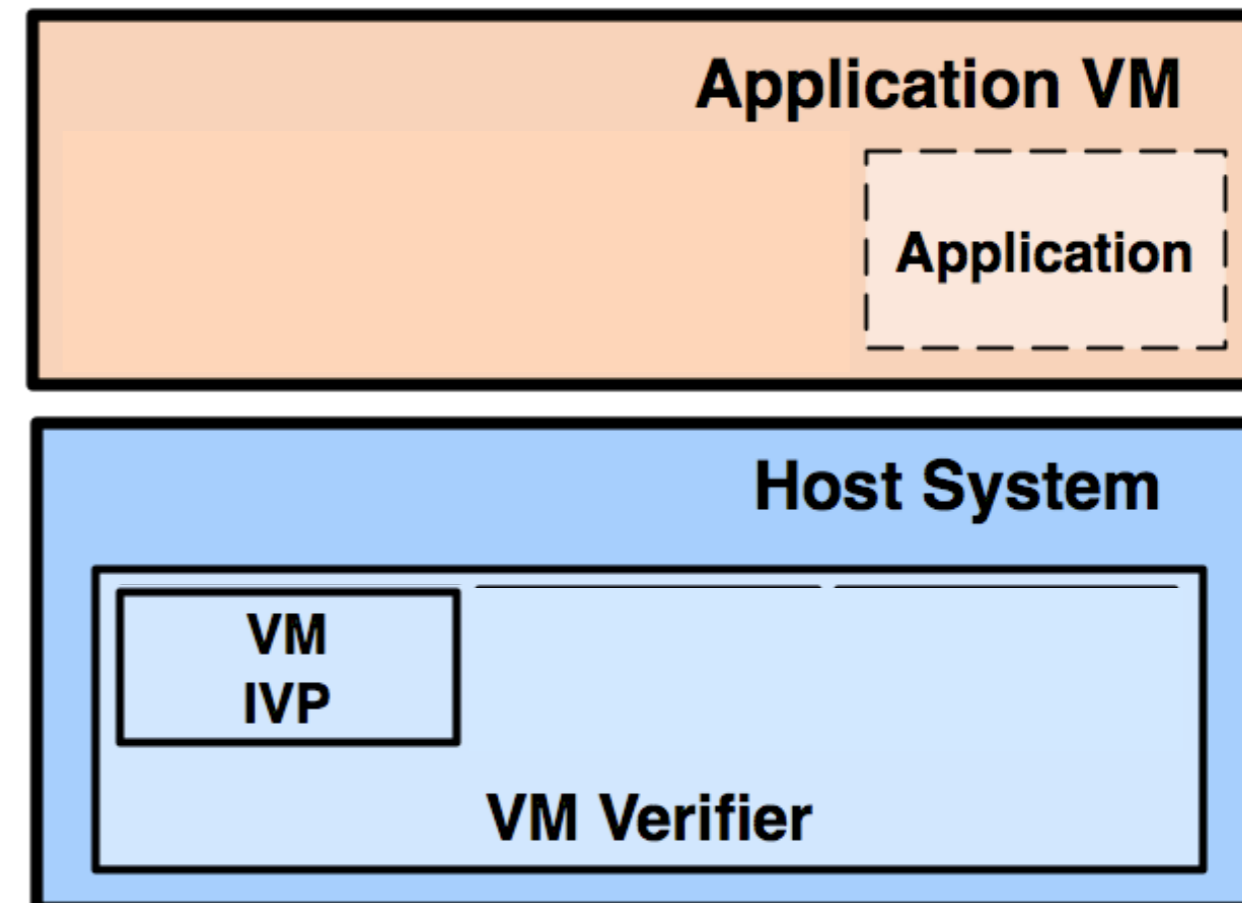
# Our Solution

- We **enforce** and enable **verification** of CW-Lite on a VM system for a particular application policy
- Simple to verify host with a VM Verifier (VMV)



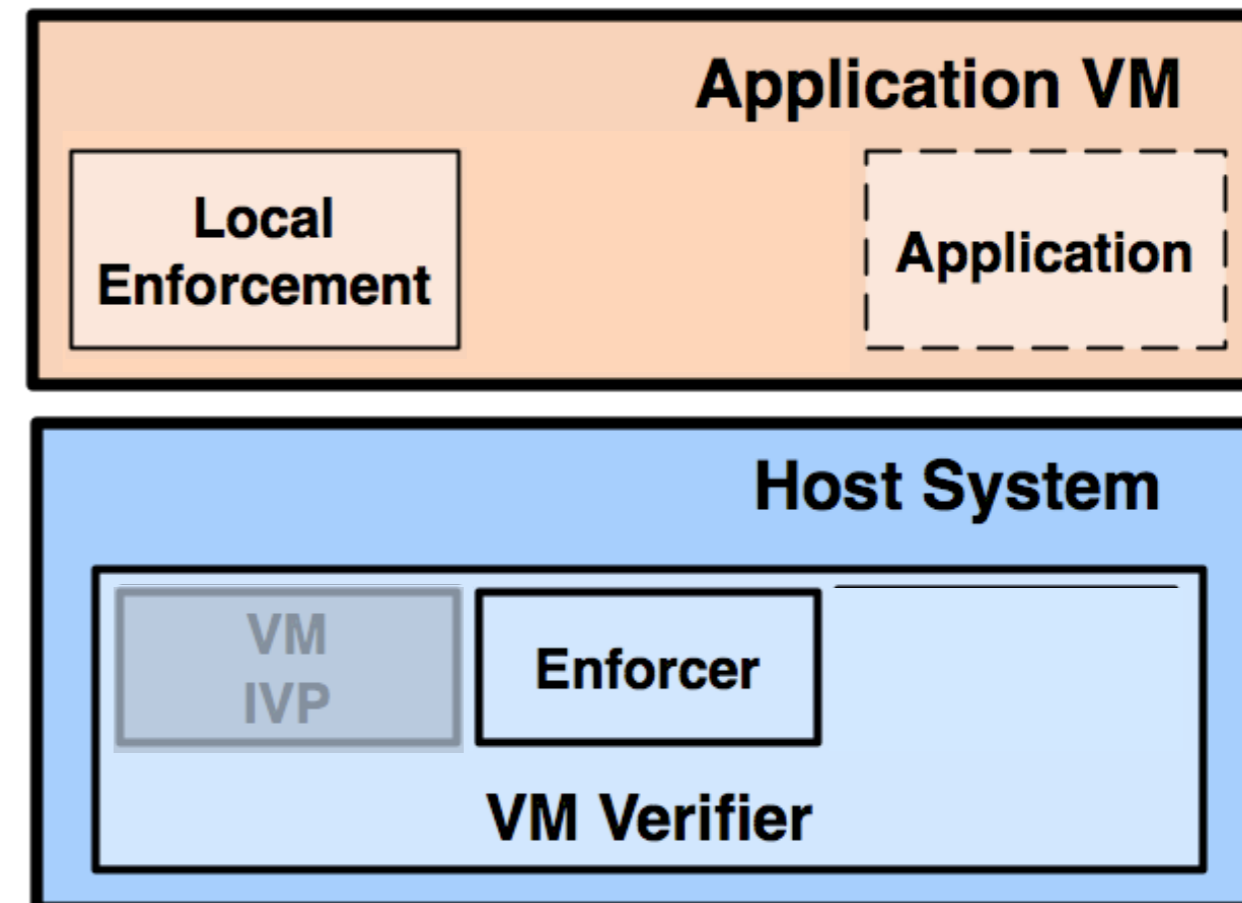
# Our Solution

- We **enforce** and enable **verification** of CW-Lite on a VM system for a particular application policy
- Simple to verify host with a VM Verifier (VMV)
  - ▶ Verify **initial integrity** of an application VM.



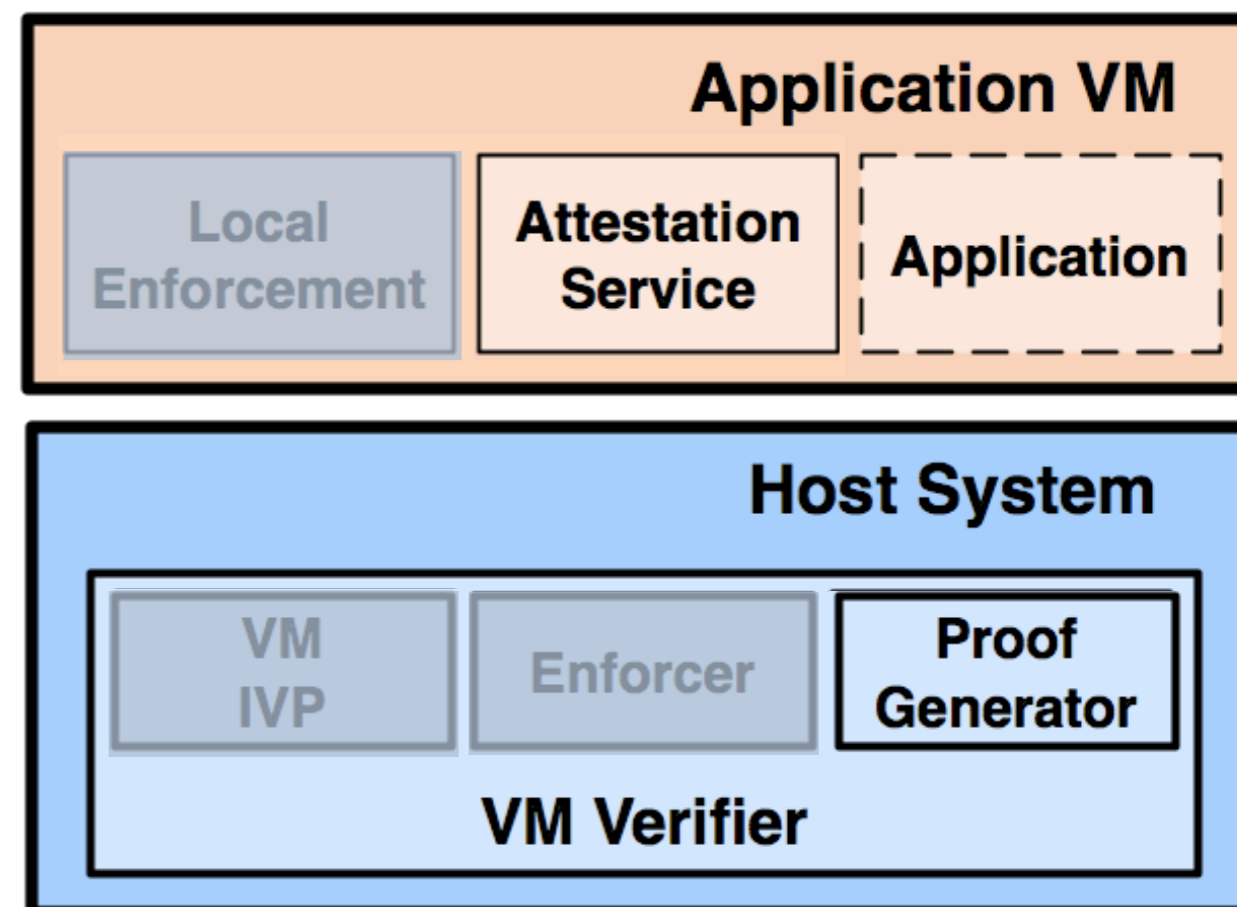
# Our Solution

- We **enforce** and enable **verification** of CW-Lite on a VM system for a particular application policy
- Simple to verify host with a VM Verifier (VMV)
  - ▶ Verify **initial integrity** of an application VM.
  - ▶ Local **enforcement** mediates integrity-relevant events



# Our Solution

- We **enforce** and enable **verification** of CW-Lite on a VM system for a particular application policy
- Simple to verify host with a VM Verifier (VMV)
  - ▶ Verify **initial integrity** of an application VM.
  - ▶ Local **enforcement** mediates integrity-relevant events
  - ▶ Attestation service handles VM **attestation** requests.



- Initial Integrity: **VM IVP** must verify VM data integrity
  - ▶ Root of trust for deploying Build VMs that satisfy CW-Lite
  - ▶ Validate Build VM integrity by inspecting an attestation of previous host system
- Enforcement: Ensuring only **trusted code** modifies Build VM data
  - ▶ Root of trust bootstraps secure execution runtime with CW-Lite policy
  - ▶ Kernel and services to enforce CW-Lite policy over Build VM execution
- Attestation: **Build integrity proof** that enables validation of CW-Lite enforcement by verifier (Build Controller)

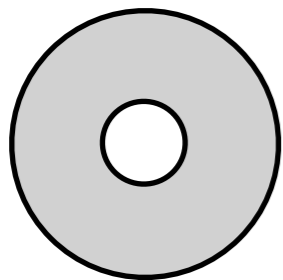
- Initial Integrity: VM IVP must verify VM data integrity
  - ▶ Root of trust for deploying Build VMs that satisfy CW-Lite
  - ▶ Validate Build VM integrity by inspecting an attestation of previous host system
- Enforcement: Ensuring only trusted code modifies Build VM data
  - ▶ Root of trust bootstraps secure execution runtime with CW-Lite policy
  - ▶ Kernel and services to enforce CW-Lite policy over Build VM execution
- Attestation: Build integrity proof that enables validation of CW-Lite enforcement by verifier (Build Controller)

# Establishing a Root of Trust

- Verification of a VM requires trust in the host
  - ▶ Rely on proper operation to protect VM integrity
  - ▶ Lack of physical hardware hampers integrity measurement
- Must measure all code and data on the host
  - ▶ Code measurement is simple to verify
  - ▶ Data is more system specific and varied

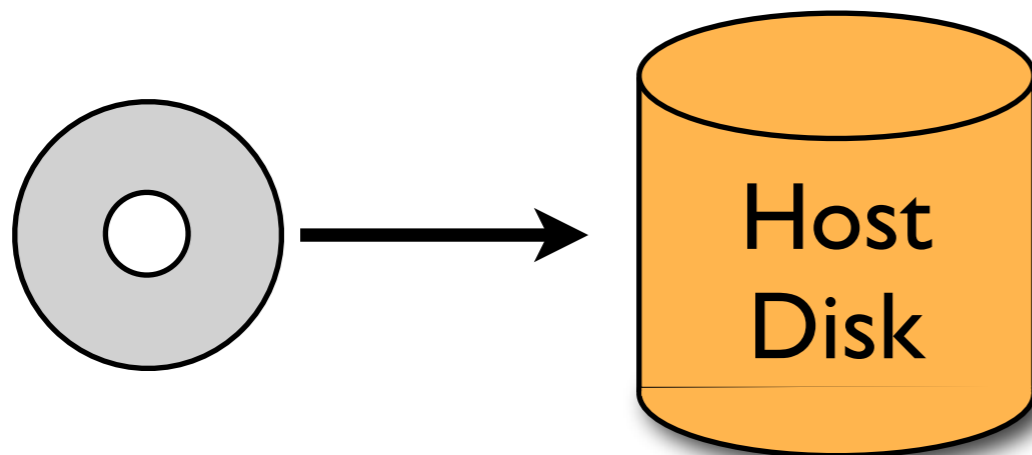


- Root of Trust for Installation (ROTI) [ACSAC '07]
  - ▶ Binds the installed filesystem to a known installer
  - ▶ Verifier can detect changes from known good state



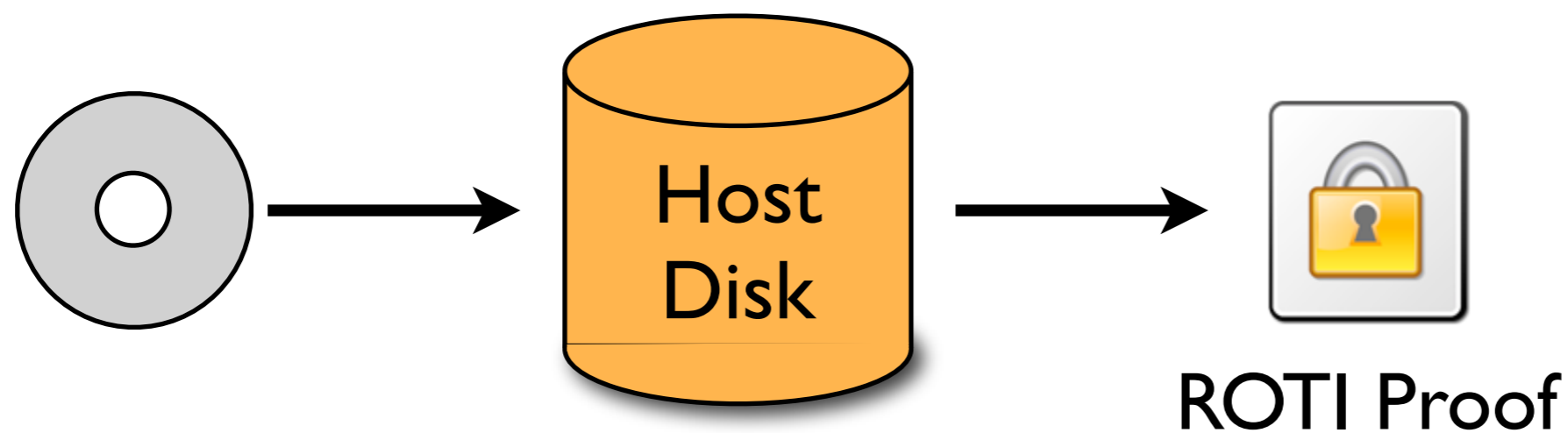
- ▶ Host then downloads VM and verifies its integrity

- Root of Trust for Installation (ROTI) [ACSAC '07]
  - ▶ Binds the installed filesystem to a known installer
  - ▶ Verifier can detect changes from known good state



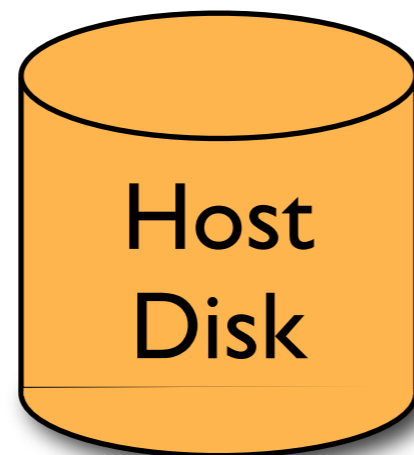
- ▶ Host then downloads VM and verifies its integrity

- Root of Trust for Installation (ROTI) [ACSAC '07]
  - ▶ Binds the installed filesystem to a known installer
  - ▶ Verifier can detect changes from known good state



- ▶ Host then downloads VM and verifies its integrity

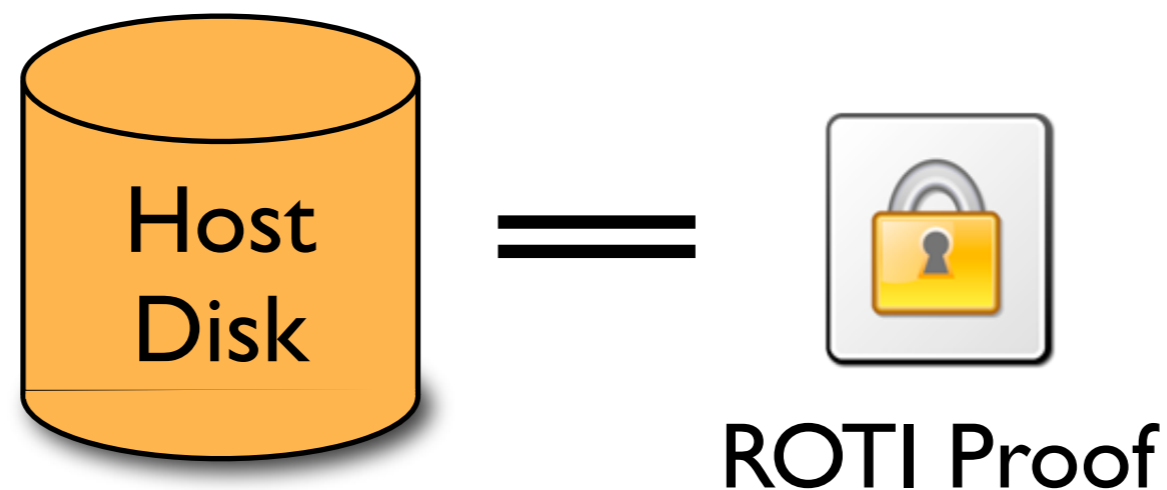
- Root of Trust for Installation (ROTI) [ACSAC '07]
  - ▶ Binds the installed filesystem to a known installer
  - ▶ Verifier can detect changes from known good state



ROTI Proof

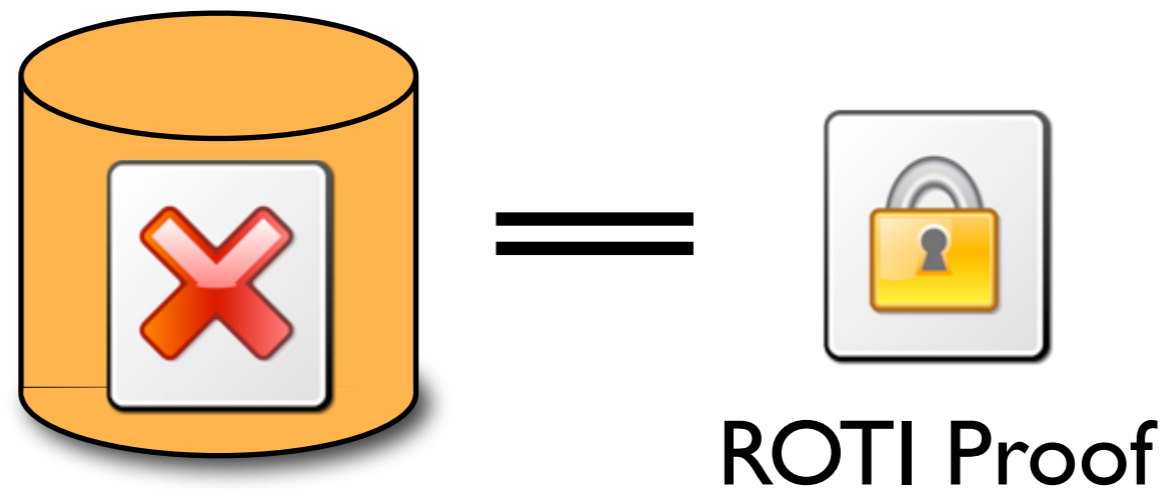
- ▶ Host then downloads VM and verifies its integrity

- Root of Trust for Installation (ROTI) [ACSAC '07]
  - ▶ Binds the installed filesystem to a known installer
  - ▶ Verifier can detect changes from known good state



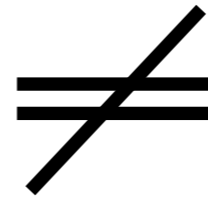
- ▶ Host then downloads VM and verifies its integrity

- Root of Trust for Installation (ROTI) [ACSAC '07]
  - ▶ Binds the installed filesystem to a known installer
  - ▶ Verifier can detect changes from known good state



- ▶ Host then downloads VM and verifies its integrity

- Root of Trust for Installation (ROTI) [ACSAC '07]
  - ▶ Binds the installed filesystem to a known installer
  - ▶ Verifier can detect changes from known good state



ROTI Proof

- ▶ Host then downloads VM and verifies its integrity

- Load runtime that enforces CW-Lite integrity
- Run only certified code (TPs)
  - ▶ Mediate loading of all executables
- Ensure that only TPs modify high integrity data (CDIs) and handle UDIs securely
  - ▶ MAC limits permissions of programs to modify CDIs
  - ▶ Interaction with remote parties must be authorized
- Two runtime components: **PRIMA Enforcement Kernel** and **Port Authority**

# PRIMA Enforcement Kernel



- We build a PRIMA enforcing kernel to mediate code loading on the VM
  - ▶ Policy defines subject labels for TPs and trusted code measurements
  - ▶ Code executed under a trusted subject label is checked before executed or mmaped for execution
  - ▶ We deny execution if code doesn't match policy database

- We build a PRIMA enforcing kernel to mediate code loading on the VM
  - ▶ Policy defines subject labels for TPs and trusted code measurements
  - ▶ Code executed under a trusted subject label is checked before executed or mmaped for execution
  - ▶ We deny execution if code doesn't match policy database
- Now we have TPs with correct code

- We build a PRIMA enforcing kernel to mediate code loading on the VM
  - ▶ Policy defines subject labels for TPs and trusted code measurements
  - ▶ Code executed under a trusted subject label is checked before executed or mmaped for execution
  - ▶ We deny execution if code doesn't match policy database
- Now we have TPs with correct code
- Next we must handle processing data items

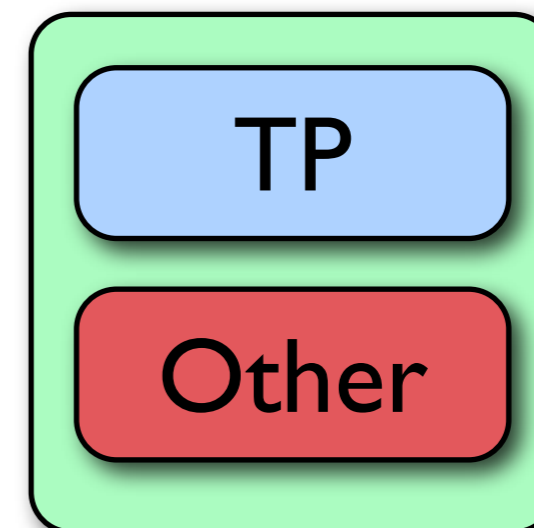
# Filtering UDI Inputs

- We can rely on local MAC enforcement to constrain UDIs through filtering interfaces
  - ▶ External data is generally considered **untrusted**
- VMV installs **PortAuthority** in the VM
  - ▶ Only permits untrusted network data through services trusted to filter

# Filtering UDI Inputs

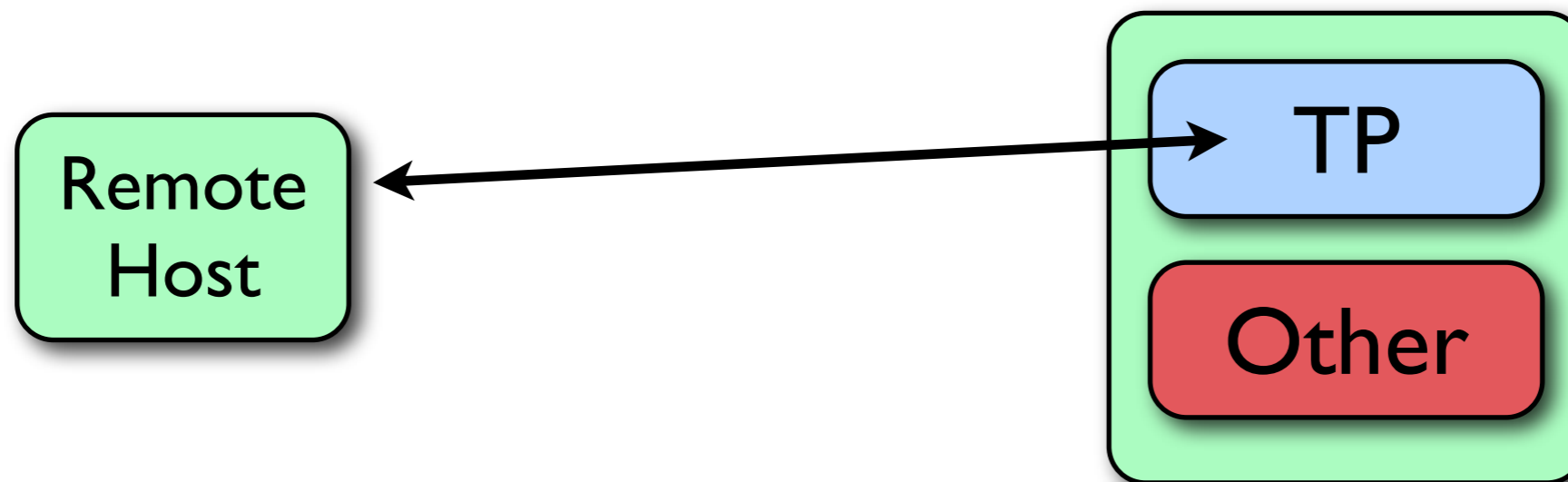
- We can rely on local MAC enforcement to constrain UDIs through filtering interfaces
  - ▶ External data is generally considered **untrusted**
- VMV installs **PortAuthority** in the VM
  - ▶ Only permits untrusted network data through services trusted to filter

Remote  
Host



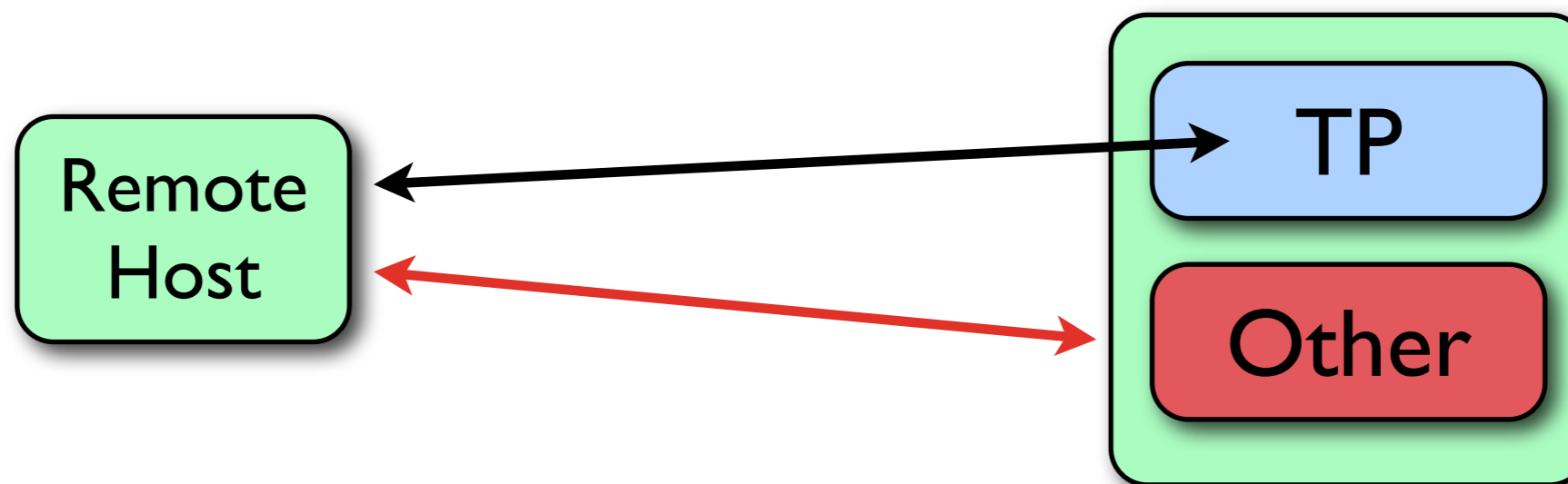
# Filtering UDI Inputs

- We can rely on local MAC enforcement to constrain UDIs through filtering interfaces
  - ▶ External data is generally considered **untrusted**
- VMV installs **PortAuthority** in the VM
  - ▶ Only permits untrusted network data through services trusted to filter



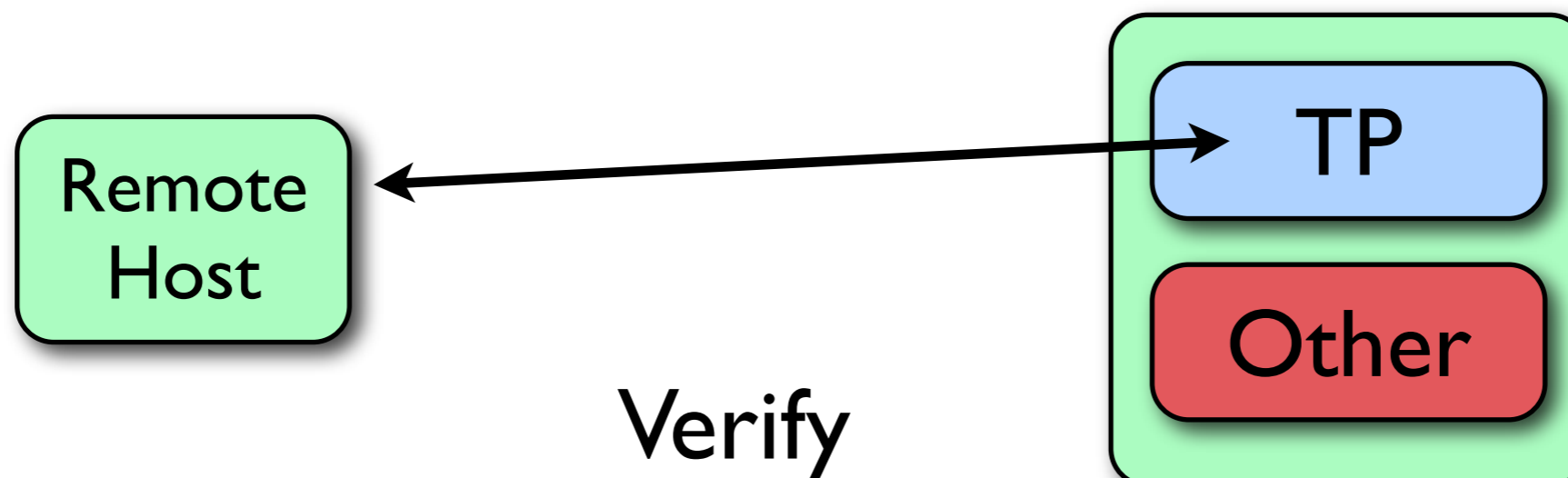
# Filtering UDI Inputs

- We can rely on local MAC enforcement to constrain UDIs through filtering interfaces
  - ▶ External data is generally considered **untrusted**
- VMV installs **PortAuthority** in the VM
  - ▶ Only permits untrusted network data through services trusted to filter



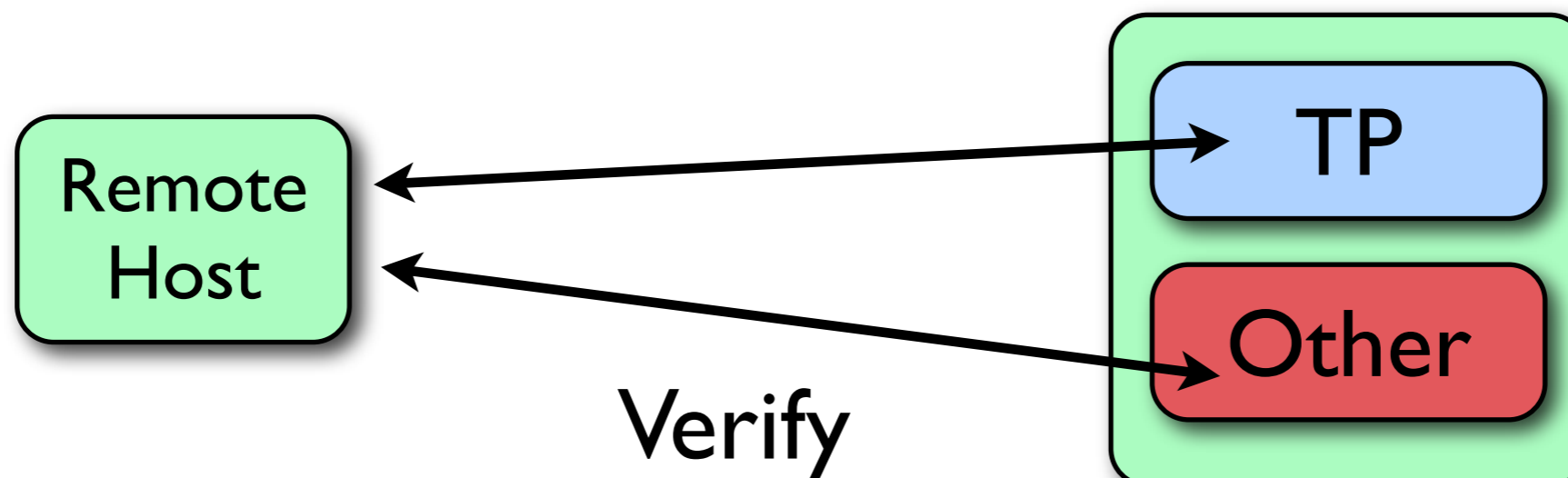
# Filtering UDI Inputs

- We can rely on local MAC enforcement to constrain UDIs through filtering interfaces
  - ▶ External data is generally considered **untrusted**
- VMV installs **PortAuthority** in the VM
  - ▶ Only permits untrusted network data through services trusted to filter



# Filtering UDI Inputs

- We can rely on local MAC enforcement to constrain UDIs through filtering interfaces
  - ▶ External data is generally considered **untrusted**
- VMV installs **PortAuthority** in the VM
  - ▶ Only permits untrusted network data through services trusted to filter



# Attesting VM Integrity

- High integrity host (ROTI) vouches for VM integrity
  - ▶ Attestation includes proof of host and application policy
- We bind the VM proof to the host platform
  - ▶ Host generates fresh public key pair for VM
  - ▶ Host TPM signs the VM's public key part
  - ▶ The remote verifier uses key to establish connection

# Attesting VM Integrity

- High integrity host (ROTI) vouches for VM integrity
  - ▶ Attestation includes proof of host and application policy
- We bind the VM proof to the host platform
  - ▶ Host generates fresh public key pair for VM
  - ▶ Host TPM signs the VM's public key part
  - ▶ The remote verifier uses key to establish connection

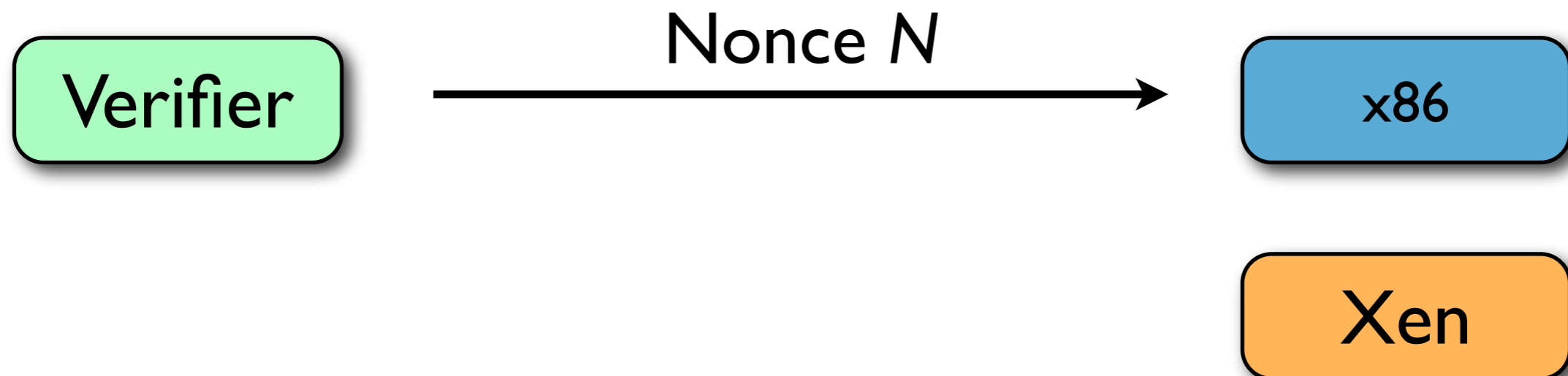
Verifier

x86

Xen

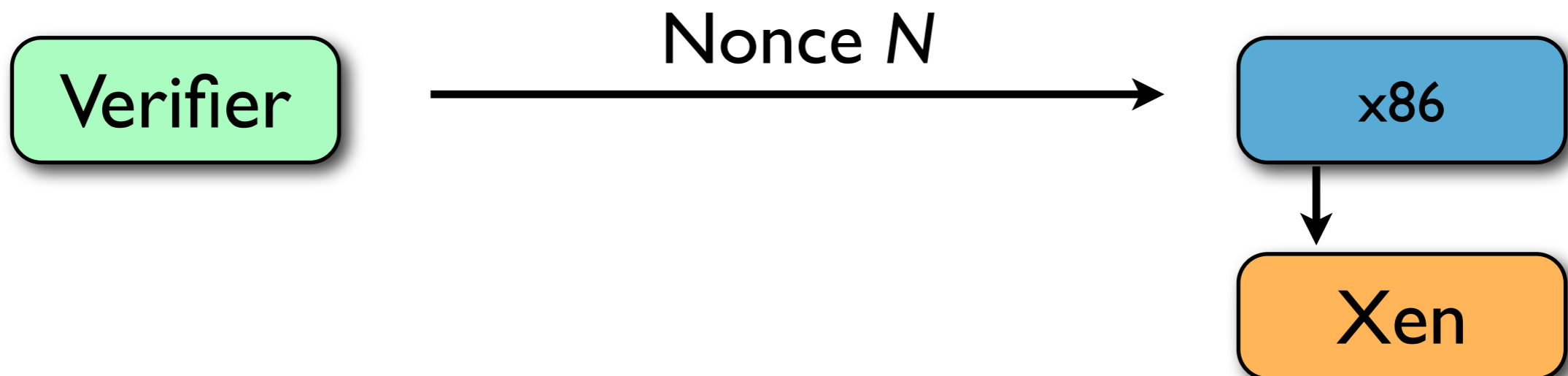
# Attesting VM Integrity

- High integrity host (ROTI) vouches for VM integrity
  - ▶ Attestation includes proof of host and application policy
- We bind the VM proof to the host platform
  - ▶ Host generates fresh public key pair for VM
  - ▶ Host TPM signs the VM's public key part
  - ▶ The remote verifier uses key to establish connection



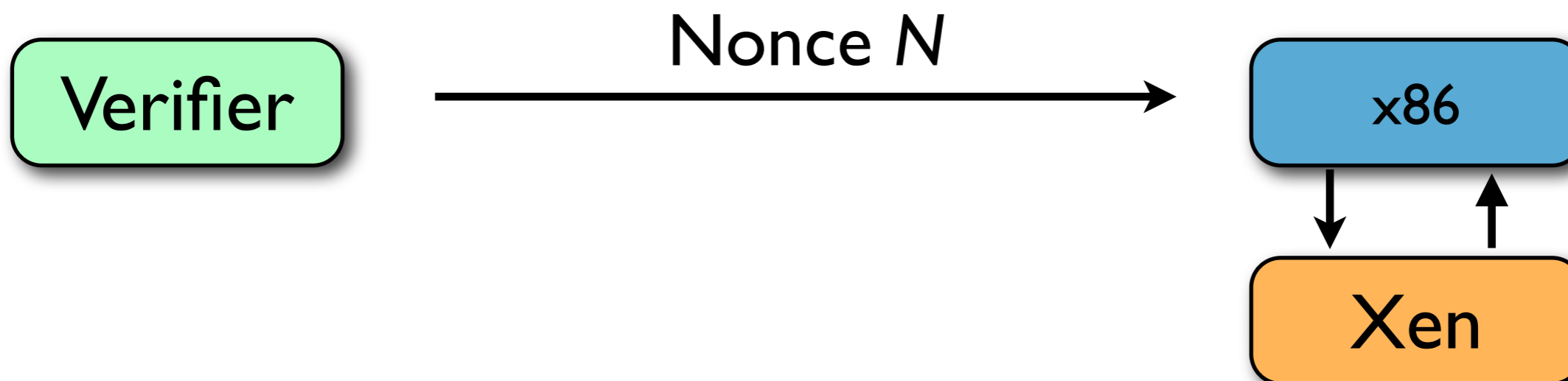
# Attesting VM Integrity

- High integrity host (ROTI) vouches for VM integrity
  - ▶ Attestation includes proof of host and application policy
- We bind the VM proof to the host platform
  - ▶ Host generates fresh public key pair for VM
  - ▶ Host TPM signs the VM's public key part
  - ▶ The remote verifier uses key to establish connection



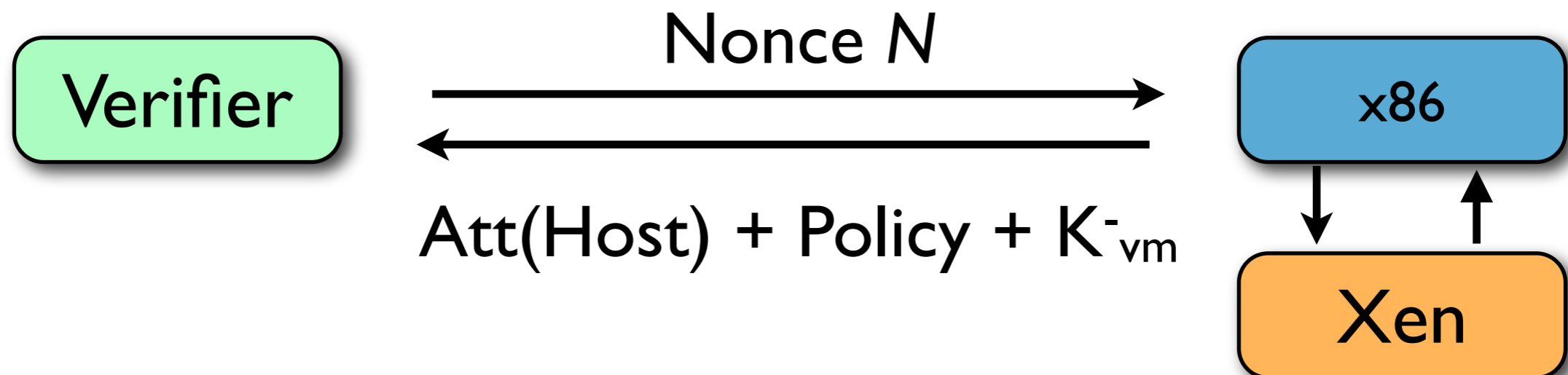
# Attesting VM Integrity

- High integrity host (ROTI) vouches for VM integrity
  - ▶ Attestation includes proof of host and application policy
- We bind the VM proof to the host platform
  - ▶ Host generates fresh public key pair for VM
  - ▶ Host TPM signs the VM's public key part
  - ▶ The remote verifier uses key to establish connection



# Attesting VM Integrity

- High integrity host (ROTI) vouches for VM integrity
  - ▶ Attestation includes proof of host and application policy
- We bind the VM proof to the host platform
  - ▶ Host generates fresh public key pair for VM
  - ▶ Host TPM signs the VM's public key part
  - ▶ The remote verifier uses key to establish connection



- We implemented our design on a distcc cluster
  - ▶ 4 Xen 3.2 host systems running 10 Ubuntu 8.04 VMs each
- Application policy
  - ▶ Ubuntu's package repository for trusted code
  - ▶ SELinux strict policy + distcc policy module
- Compiled several code bases
  - ▶ Found about a 4% overhead in compilation time

Task	Time (mins.)		% diff.	Lines of Code
	Unattested	Attested		
Linux Kernel	7:01.84	7:17.24	3.65%	6,450,761
OpenSSH	0:22.67	0:23.74	3.98%	68,626

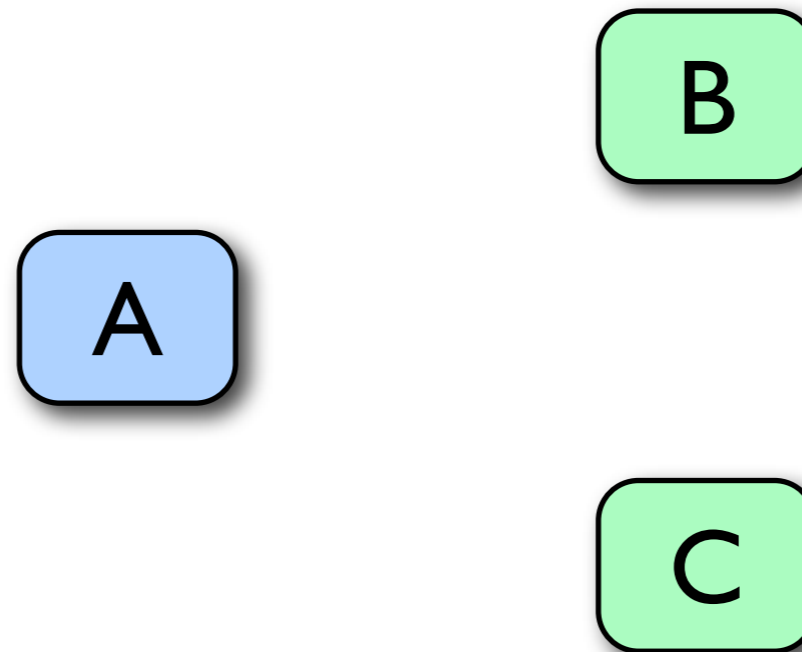
# Considerations

# Considerations

- Runtime code integrity checks
- Associating proofs with the application results
- Transitivity among components

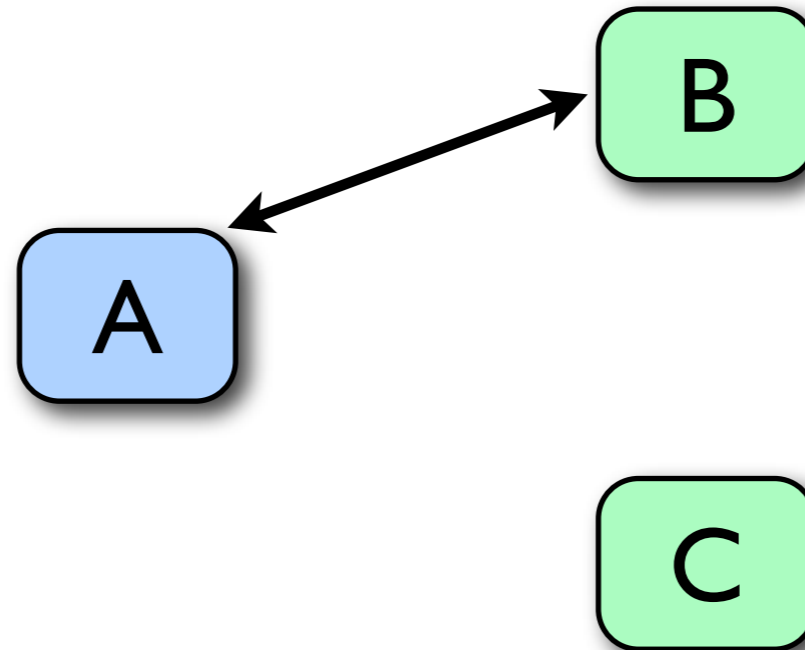
# Considerations

- Runtime code integrity checks
- Associating proofs with the application results
- Transitivity among components



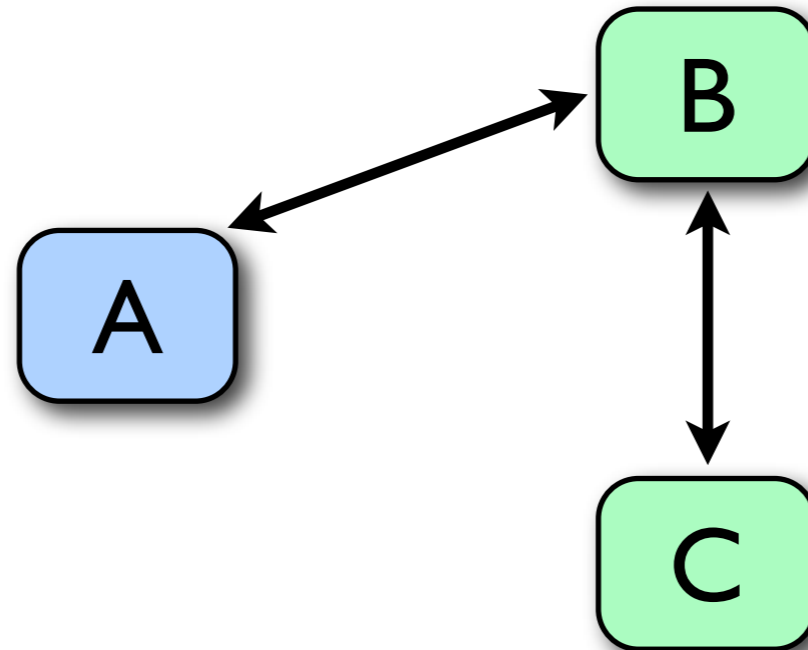
# Considerations

- Runtime code integrity checks
- Associating proofs with the application results
- Transitivity among components



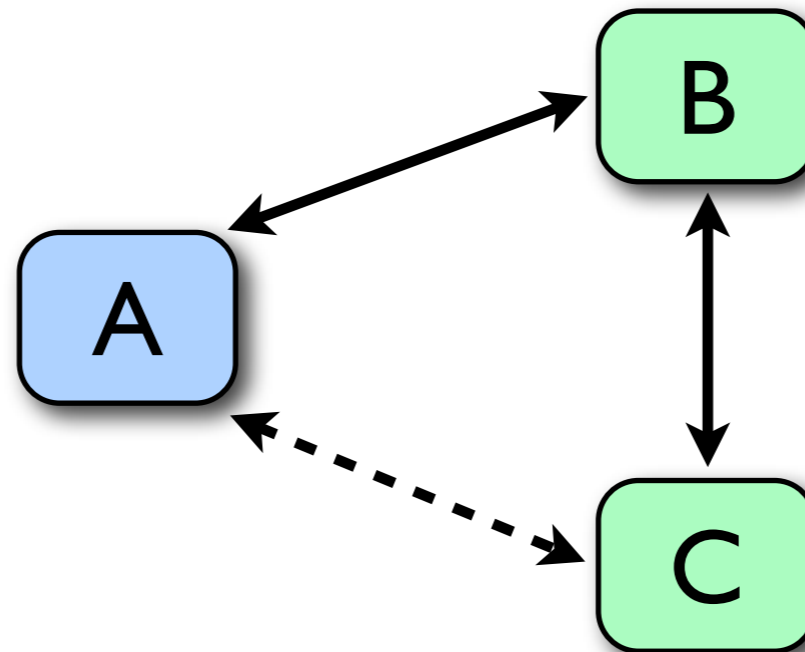
# Considerations

- Runtime code integrity checks
- Associating proofs with the application results
- Transitivity among components

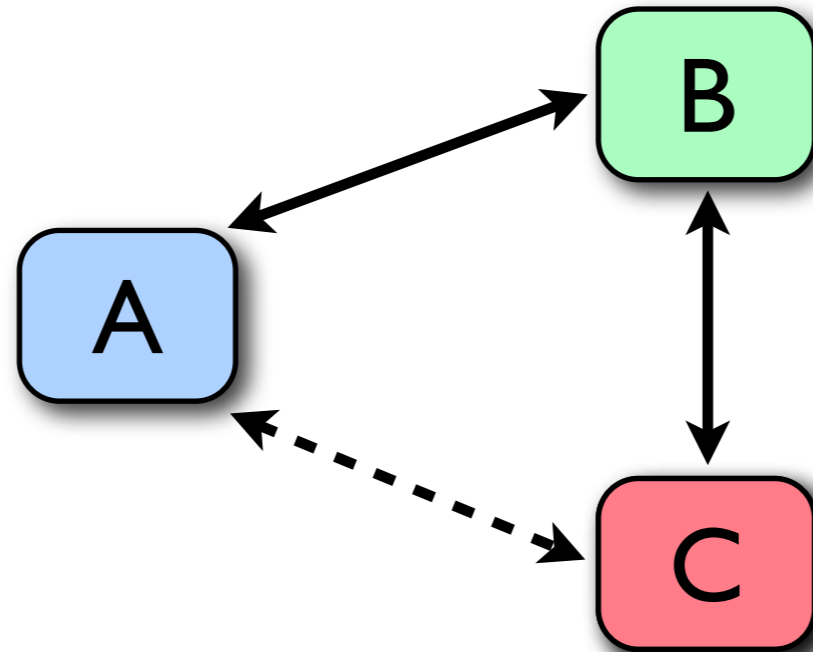


# Considerations

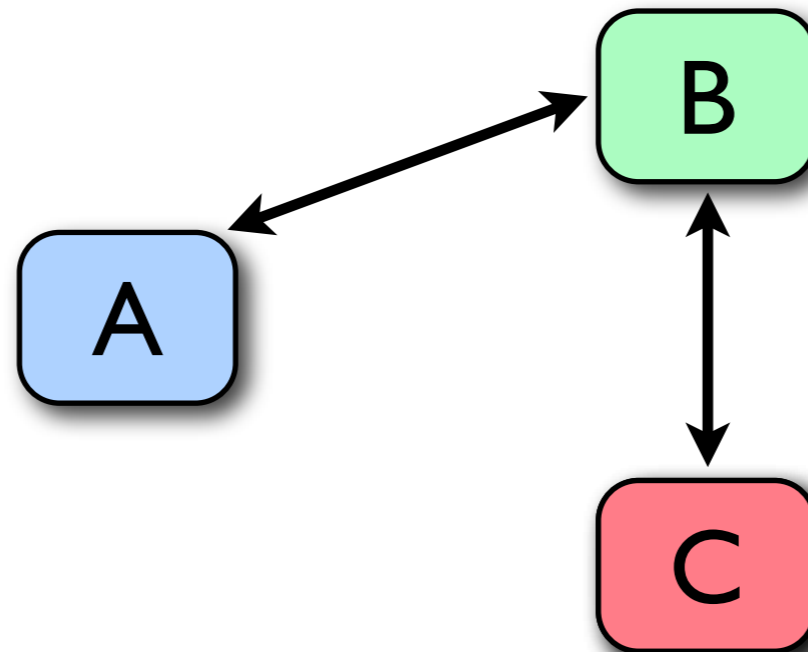
- Runtime code integrity checks
- Associating proofs with the application results
- Transitivity among components



- Runtime code integrity checks
- Associating proofs with the application results
- Transitivity among components



- Runtime code integrity checks
- Associating proofs with the application results
- Transitivity among components



- Our approach supports comprehensive system integrity enforcement and verification on VM-based distributed computing nodes
  - ▶ Pull together various integrity measurement ideas
  - ▶ Into a complete architecture for high integrity execution
- Introduces only a minimal overhead on our proof of concept distributed compilation cluster
- Extensible to various integrity policies and enforcement mechanisms

# Questions?

Joshua Schiffman ([jschiffm@cse.psu.edu](mailto:jschiffm@cse.psu.edu))

<http://www.joshschiffman.org/>

*SIS Laboratory* (<http://sis.cse.psu.edu>)