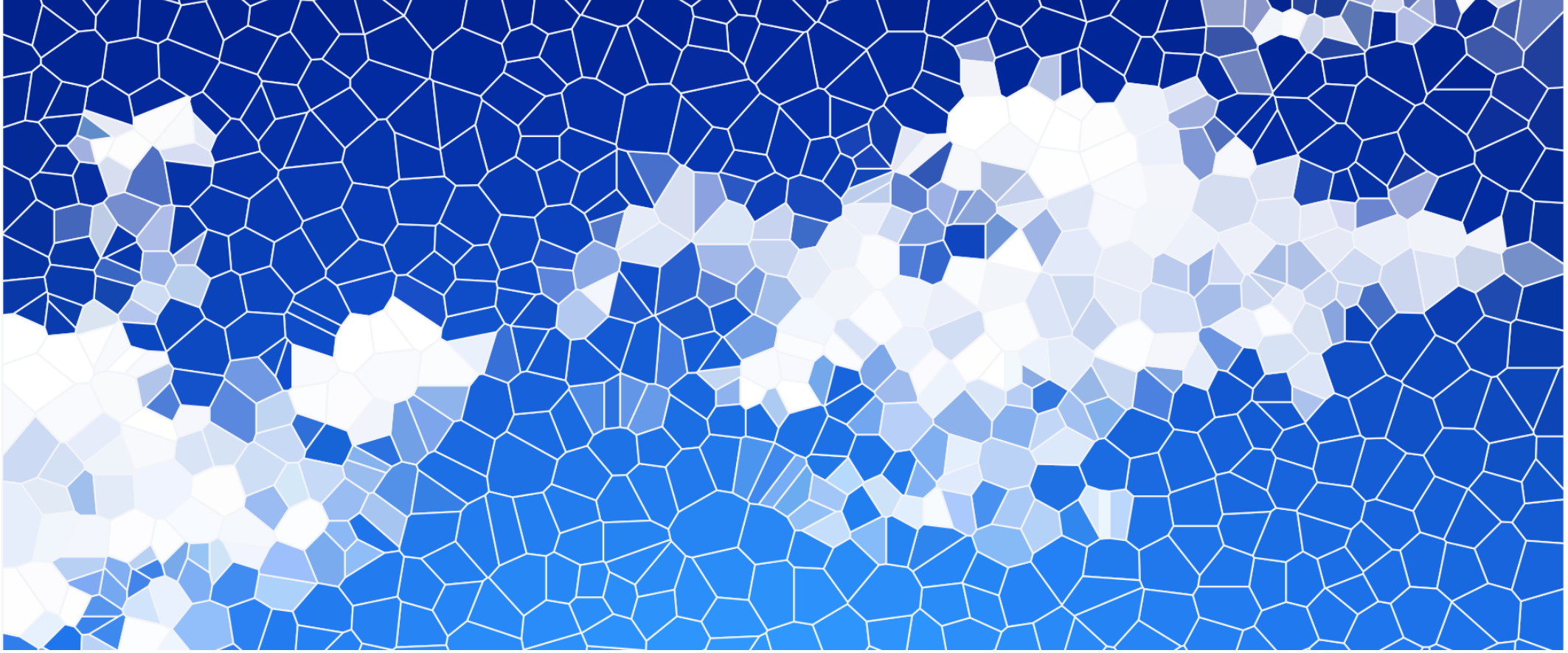


SCONE – SECURE CONTAINERS

OSDI2016, EuroSys2017

Christof Fetzer, TU Dresden, Germany

MOTIVATION

- **Role: Service Provider**

- **Data is valuable,** we need to protect
  - **confidentiality,** and
  - **integrity**

THREAT MODEL

- not trusting cloud nor development machines -

THREAT MODEL (PARTIAL)

- **System administrator not trusted**
  - **but** system administrators have root access and
  - e.g., can dump process main memory with all keys

- We **cannot trust**
  - **integrity / confidentiality of input nor output**

EXECUTIVE SUMMARY

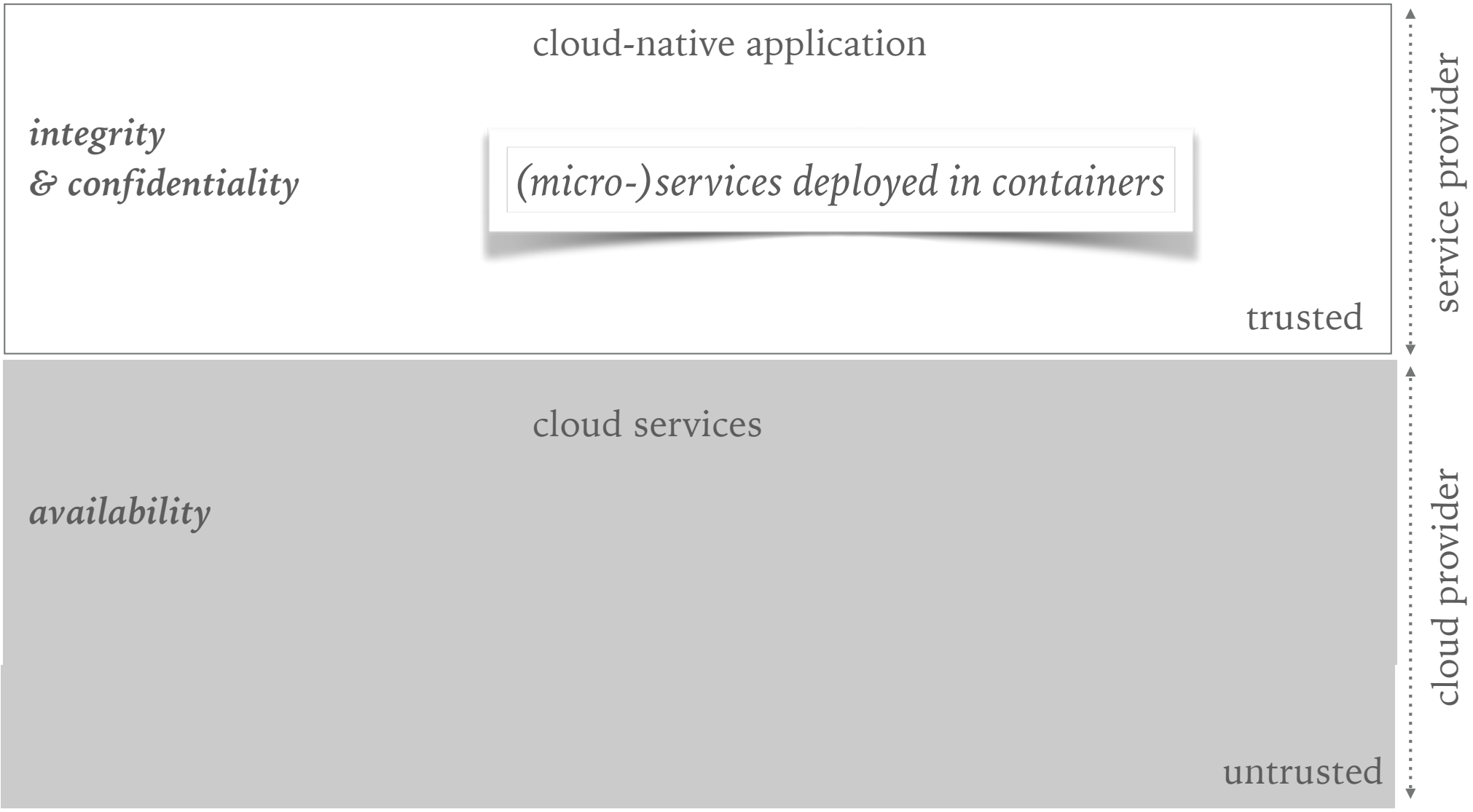
- **SCONE platform:**
  - simplifies running applications in Intel SGX enclaves
  - focus on **ease of use**
    - transparent attestation and configuration
    - no application code changes


```
> my_app
```

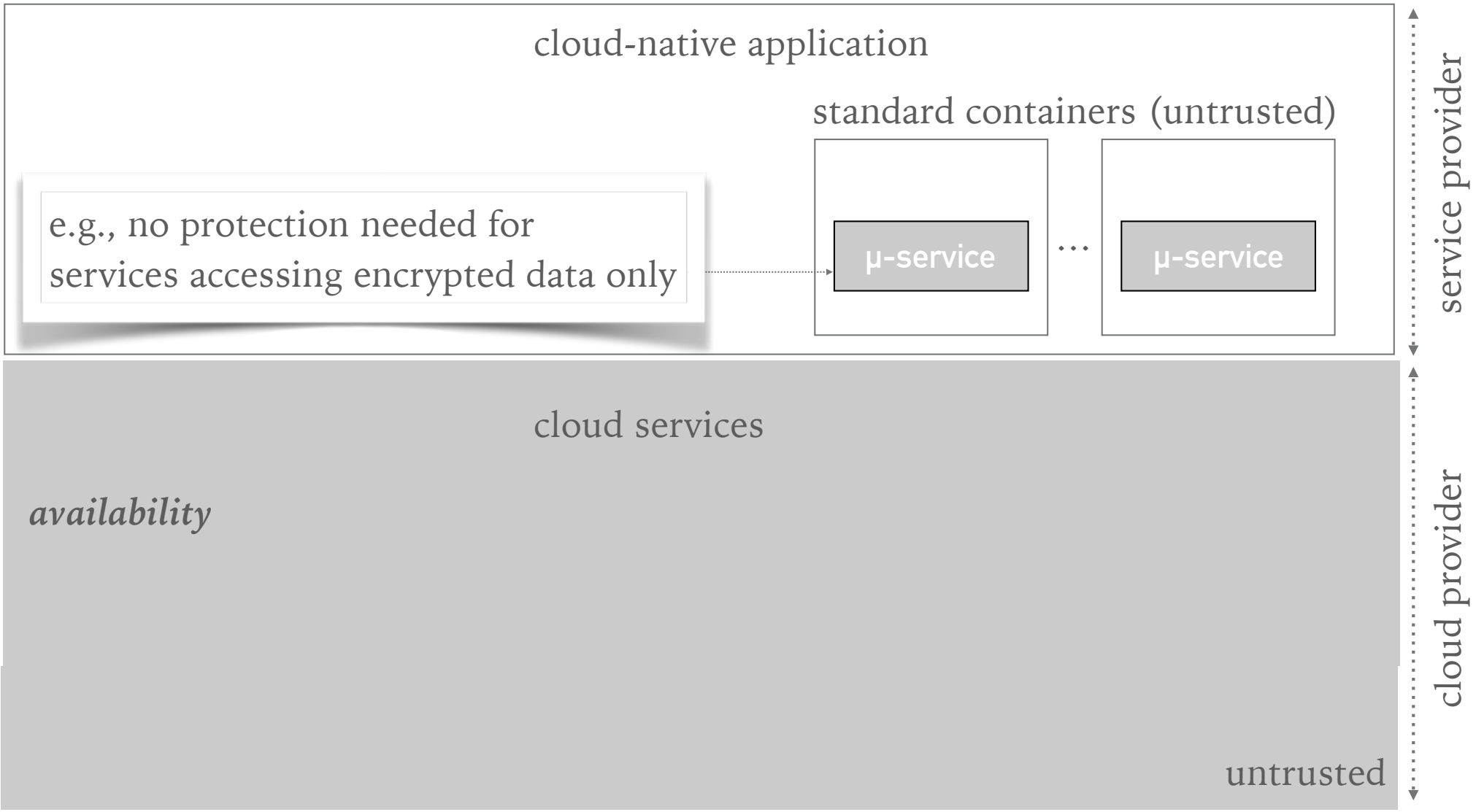
```
> SCONE_ALPINE=1 my_app
```

SCONE GENERAL APPROACH

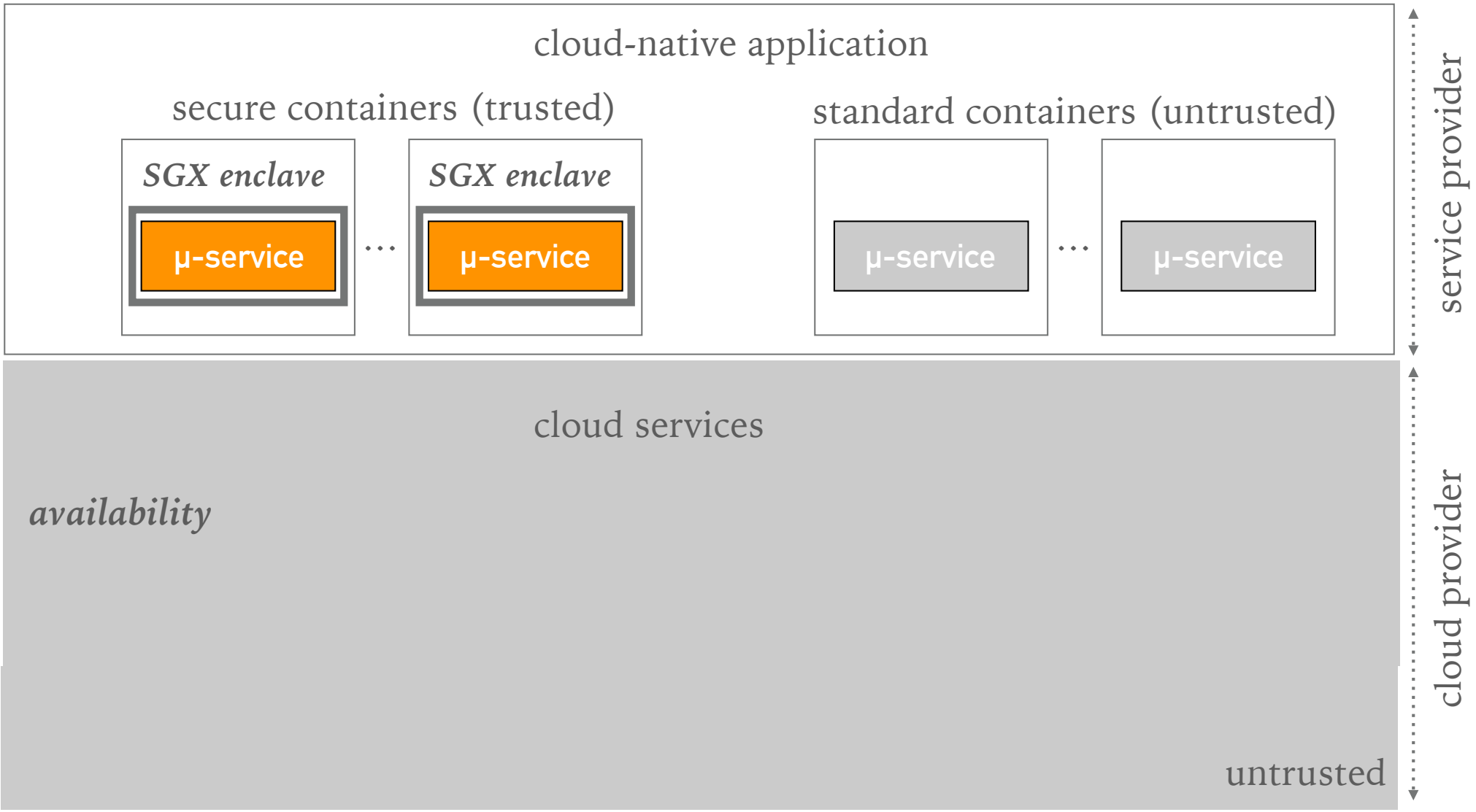
DIVIDE AND CONQUER



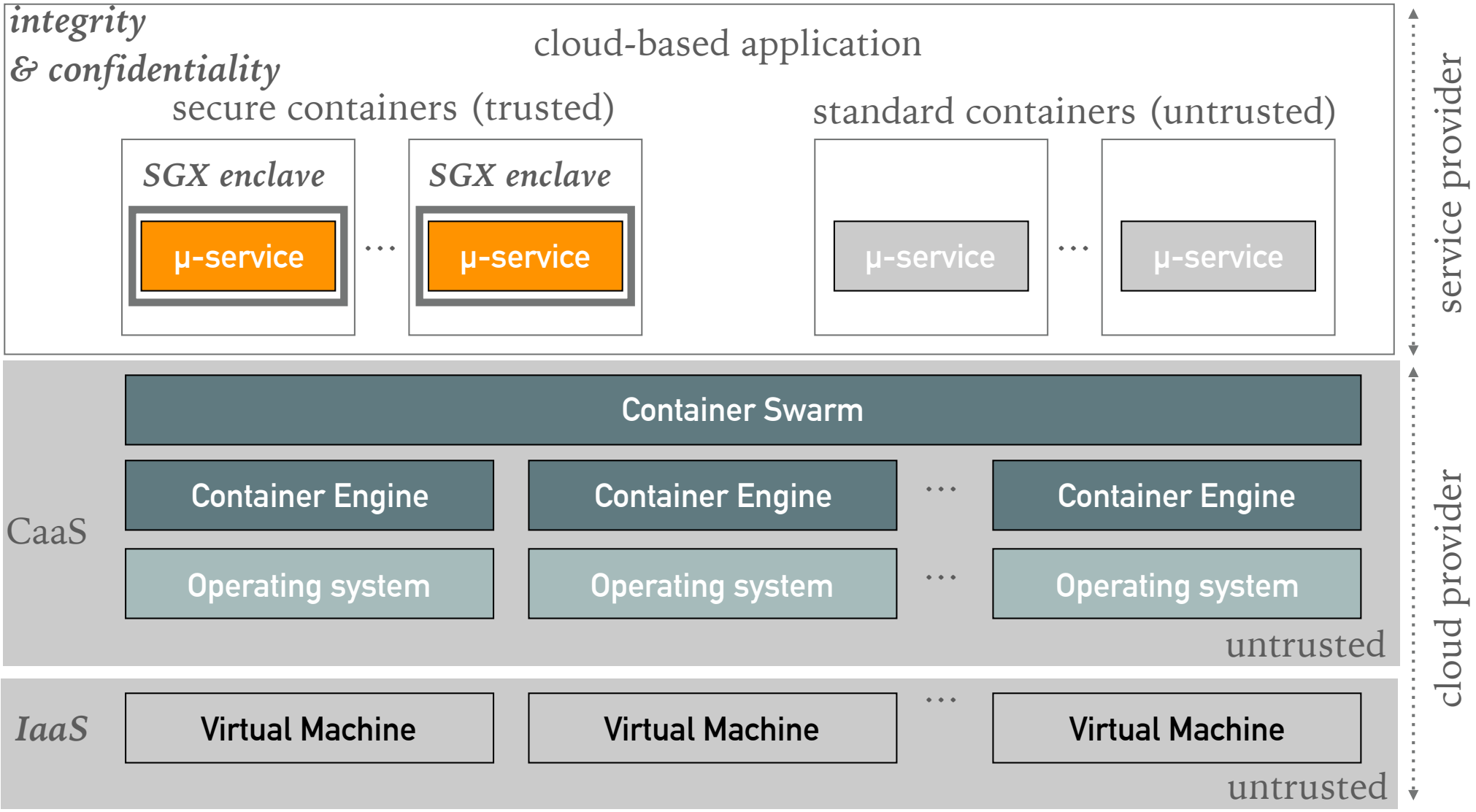
EACH MICROSERVICE RUNS IN A CONTAINER



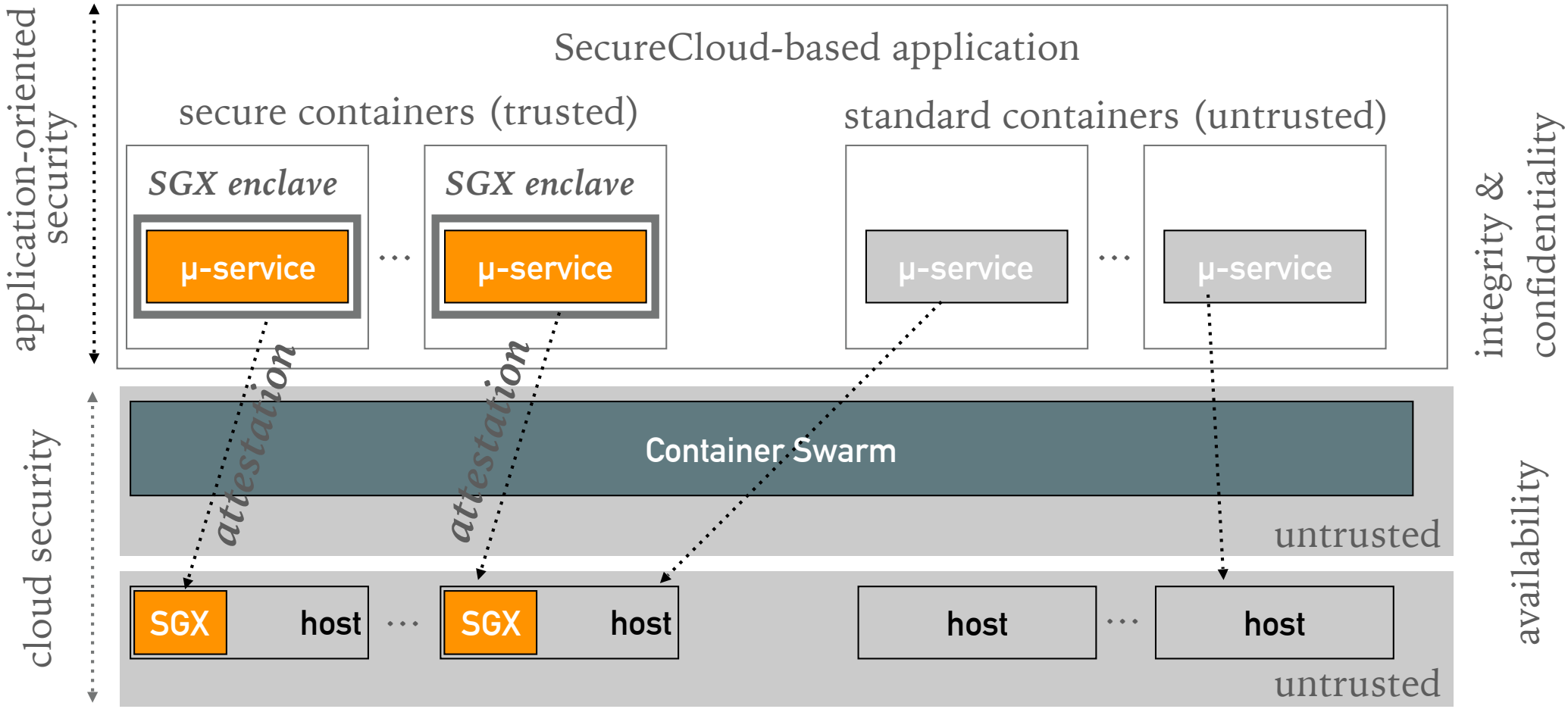
TOP-LEVEL ARCHITECTURE



CAAS, IAAS (OR MAAS)



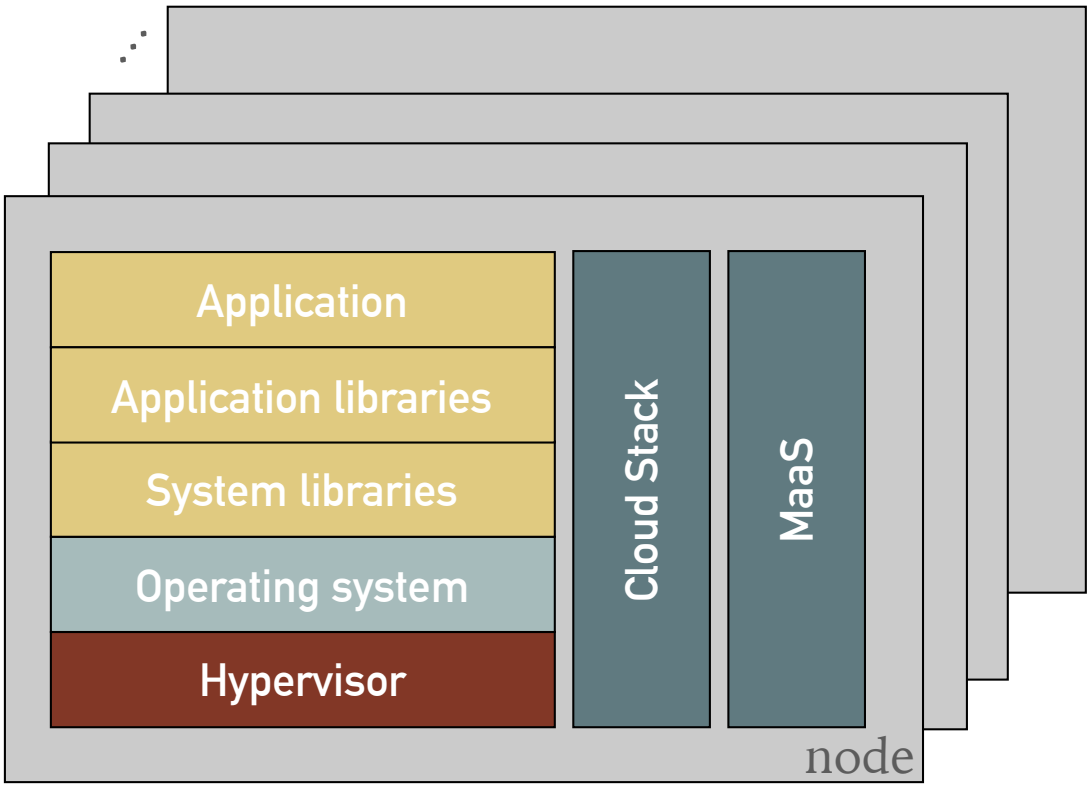
APPLICATION-ORIENTED SECURITY VS CLOUD SECURITY



TO PARTITION OR NOT TO PARTITION

- single processes -

DEFENDER'S DILEMMA



cloud software stack

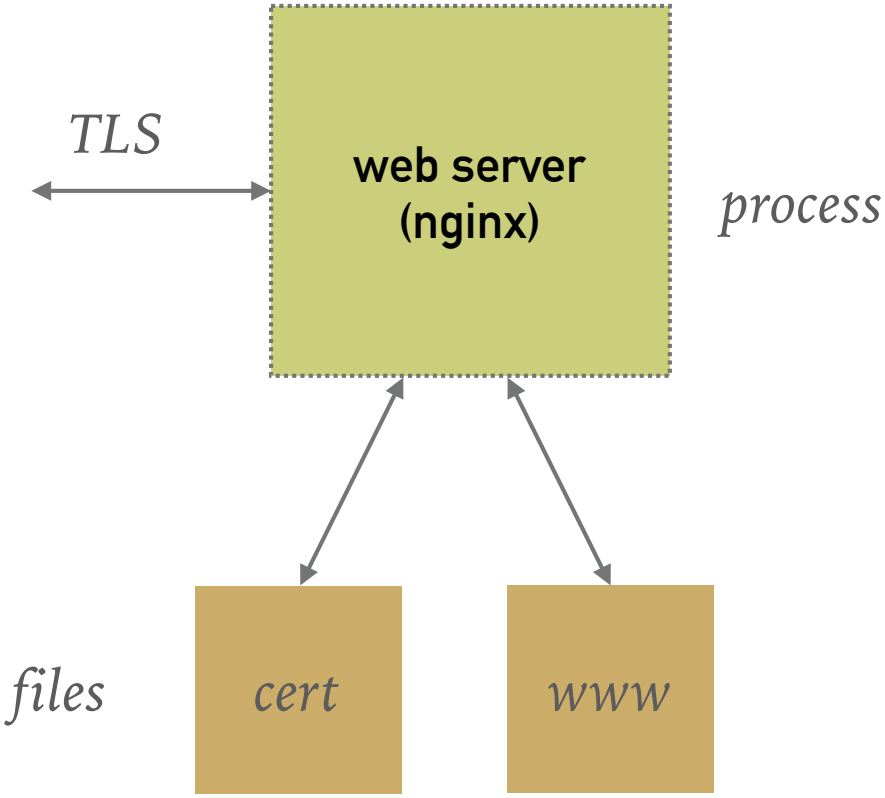
- **Attackers**:
  - success by exploiting a single vulnerability
- Defender:
  - must protect against every vulnerability
    - **not only in application**
  - millions of lines of source code

VULNERABILITIES

- **Coverity reports:**
  - 1 defect per 1700 lines of code
- **Kernel self protection project:**
  - 500 security bugs fixed in Linux during the last 5 years
  - each bug stayed about 5 years inside kernel
- **Coverity:**
  - quality of closed source software is not better than open source software
- **MacOS**: no root password needed

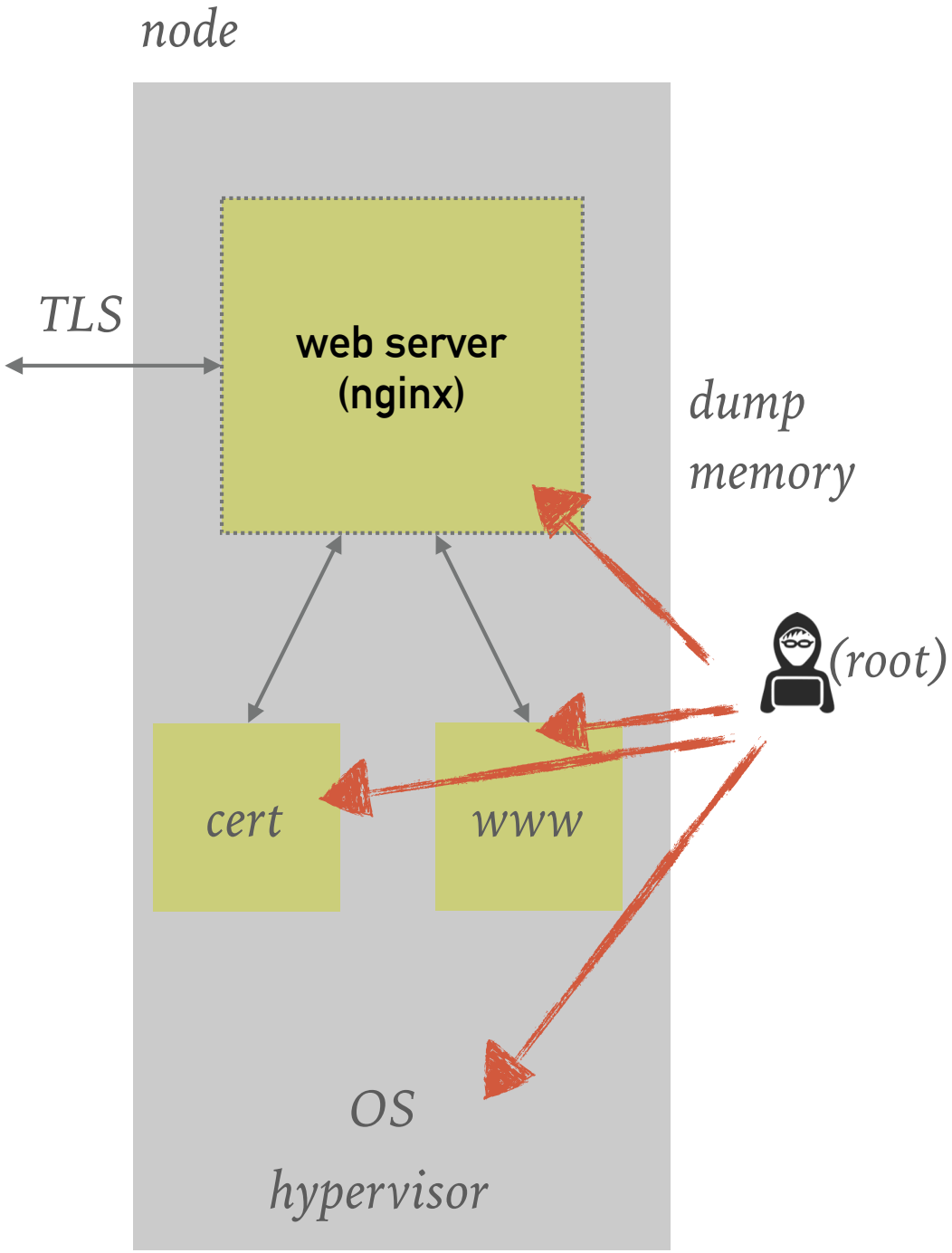
[KSPP] Kees Cook, The State of Kernel Self Protection Project, Linux Security Summit (LSS), 2016

[Coverity] Open Source Report 2014 - Coverity, go.coverity.com/rs/157-LQW.../2014-Coverity-Scan-Report.pdf



EXAMPLE

- Web Server (**nginx**)
- **Configuration:**
  - TLS certificate (private key)
  - config file
  - …
- **WWW files:**
  - must only visible to authorised clients

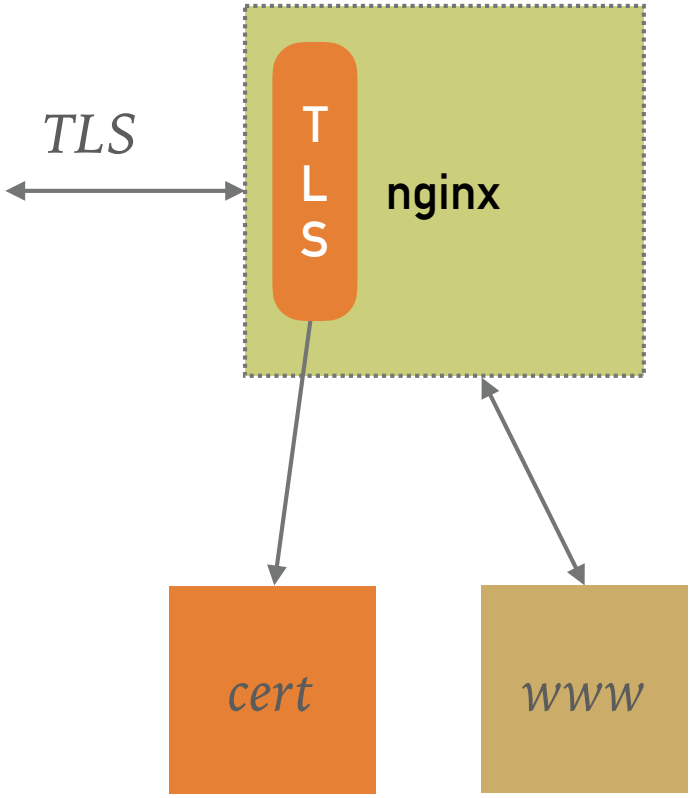


THREAT MODEL?

- Attacker has root access
  - controls OS
  - controls Hypervisor
- **Attacker can**
  - read/modify all files
  - can read/modify memory of processes
  - can see all network traffic

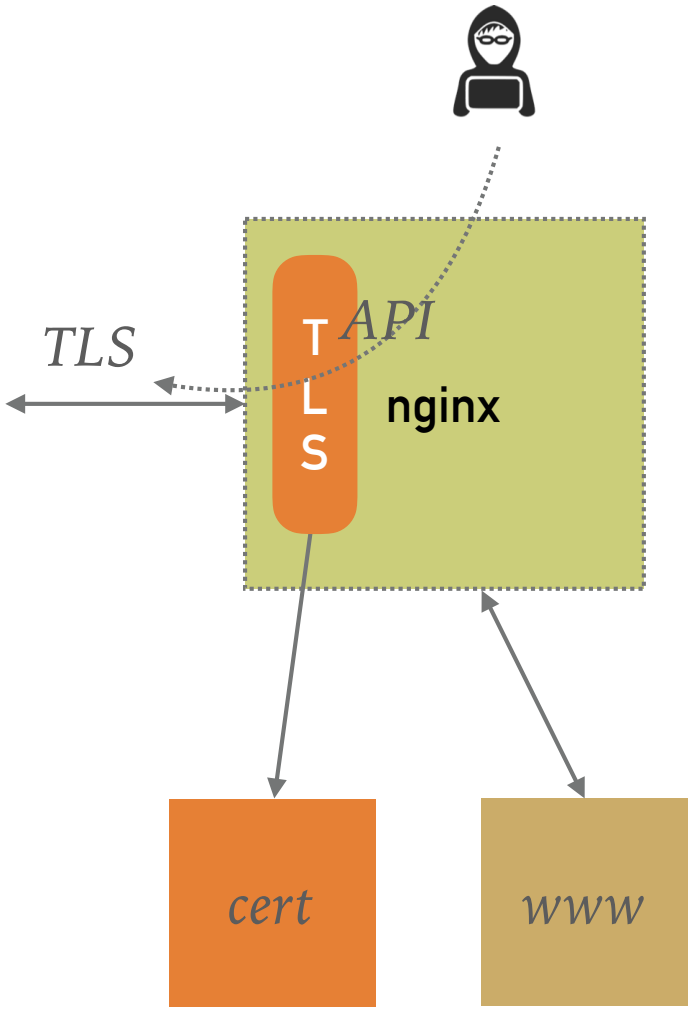
SHOULD WE PARTITION NGINX?

- We need to protect certificate!
  - must not leak
  - TLS should be protected!



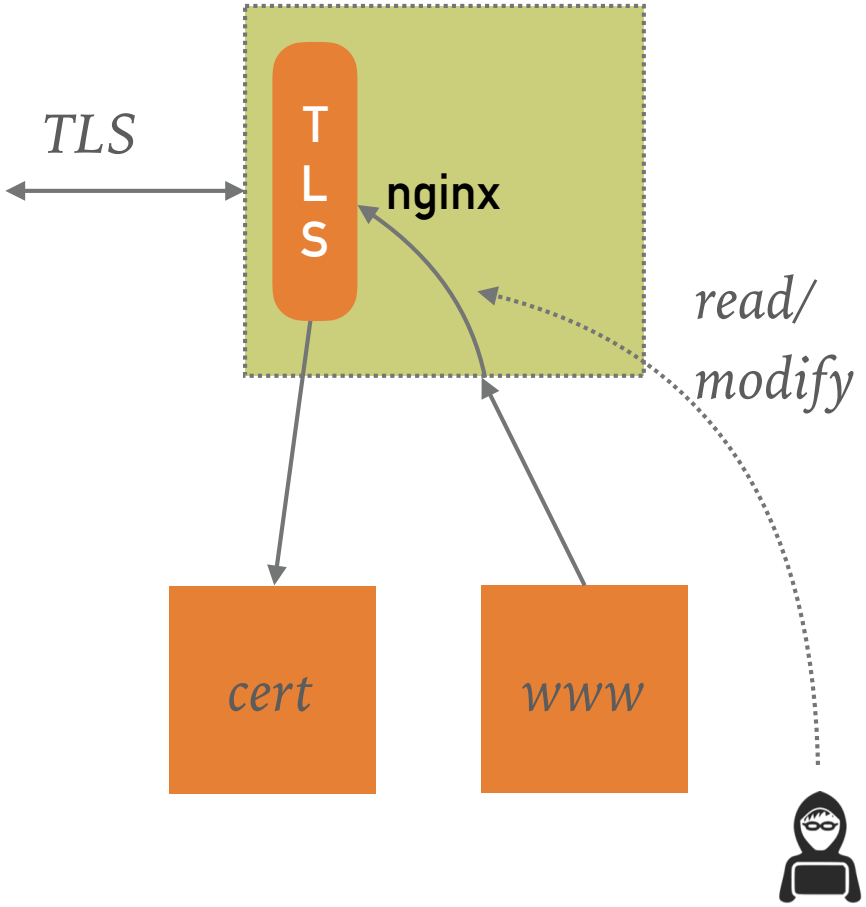
could impersonate original website if not protected

[Glamdring, Usenix ATC 2017]



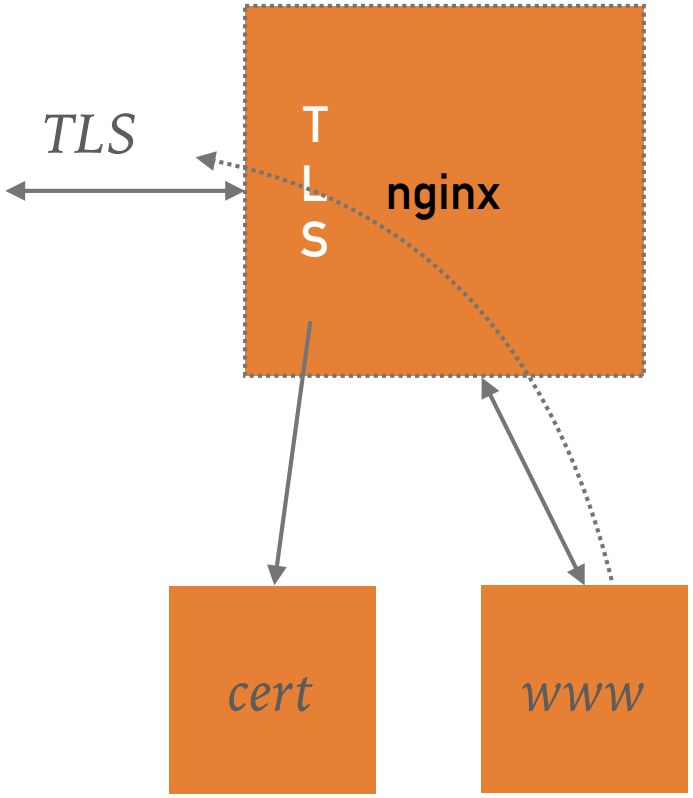
SHOULD WE PARTITION NGINX?

- We need to protect certificate!
  - must not leak
  - TLS should be protected!
- **Attacker does not need cert:**
  - **establish connections via protected TLS stack**
- how to protect against this?
  - how to automate the protection?



SHOULD WE PARTITION NGINX?

- We need to protect certificate!
  - must not leak
  - TLS should be protected!
- We need to encrypt www files
  - to ensure confidentiality
  - to ensure integrity



SHOULD WE PARTITION NGINX?

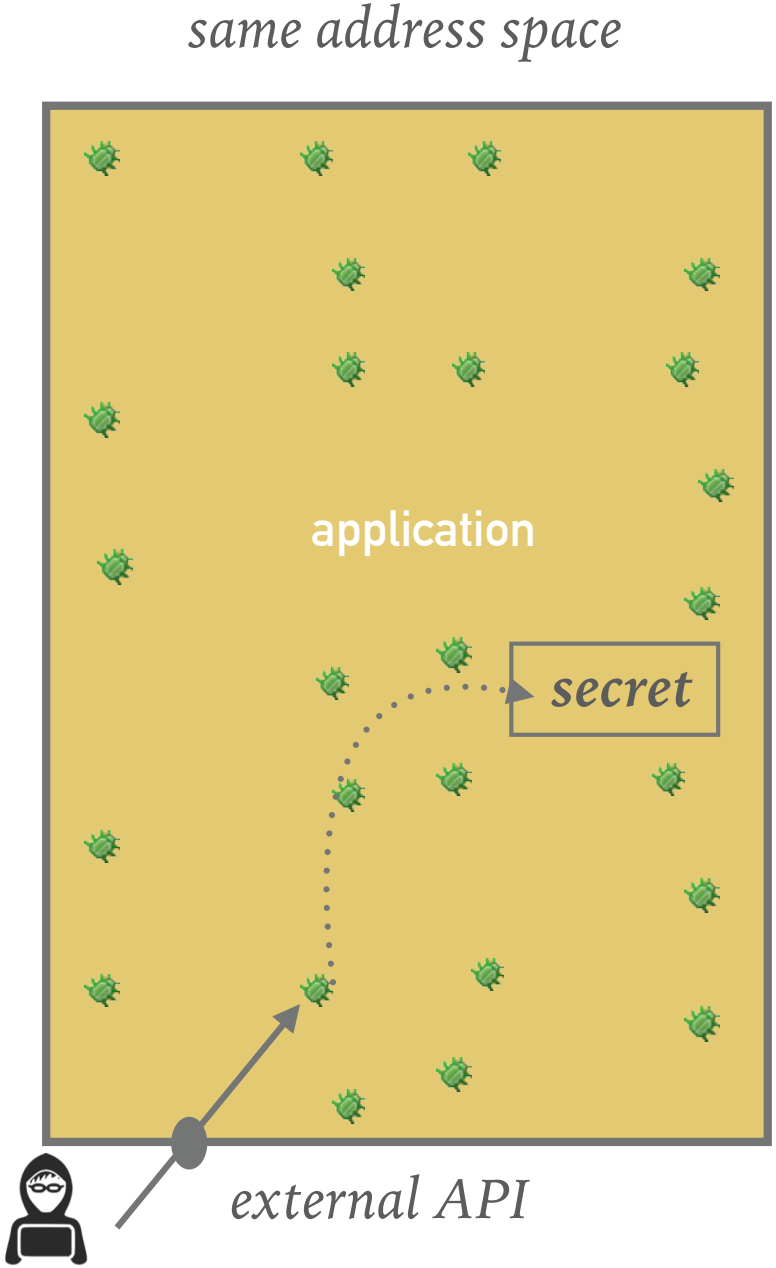
- We need to protect certificate!
  - must not leak
  - TLS should be protected!
- We need to encrypt www files
  - to ensure confidentiality
  - to ensure integrity
- We need to protect content
  - never as plain text
  - detect modifications

HOW TO ATTACK SCONE-BASED APPLICATIONS?

- **via OS interface:** *[Scone, OSDI 2016]*
  - **SCONE provides standard shields** (reuse across applications)
- **via side channels:** *[under submission]*
  - SCONE is resistant against side channel attacks
- **via software bugs in application:**
  - make it difficult to exploit

SOFTWARE BUGS

- **Bounds checker (SGXBounds):** *[Scone, EuroSys 2017]*
  - protect against low-level vulnerabilities
  - e.g., protects against Heartbleed
- **Focus on microservices:** *[IEEE Sec & Priv., 2016]*
  - isolation of microservices
- **Protect against triggering software bugs:**
  - by limiting access to APIs of interfaces

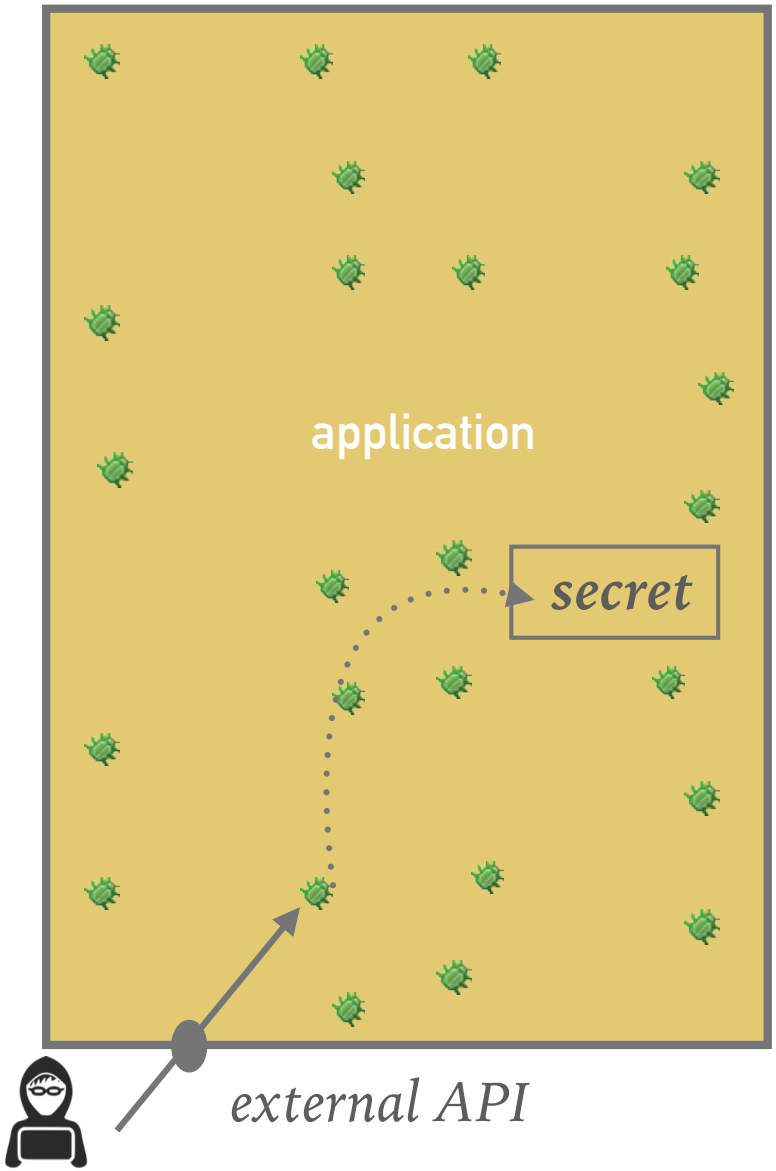


SOFTWARE BUGS!

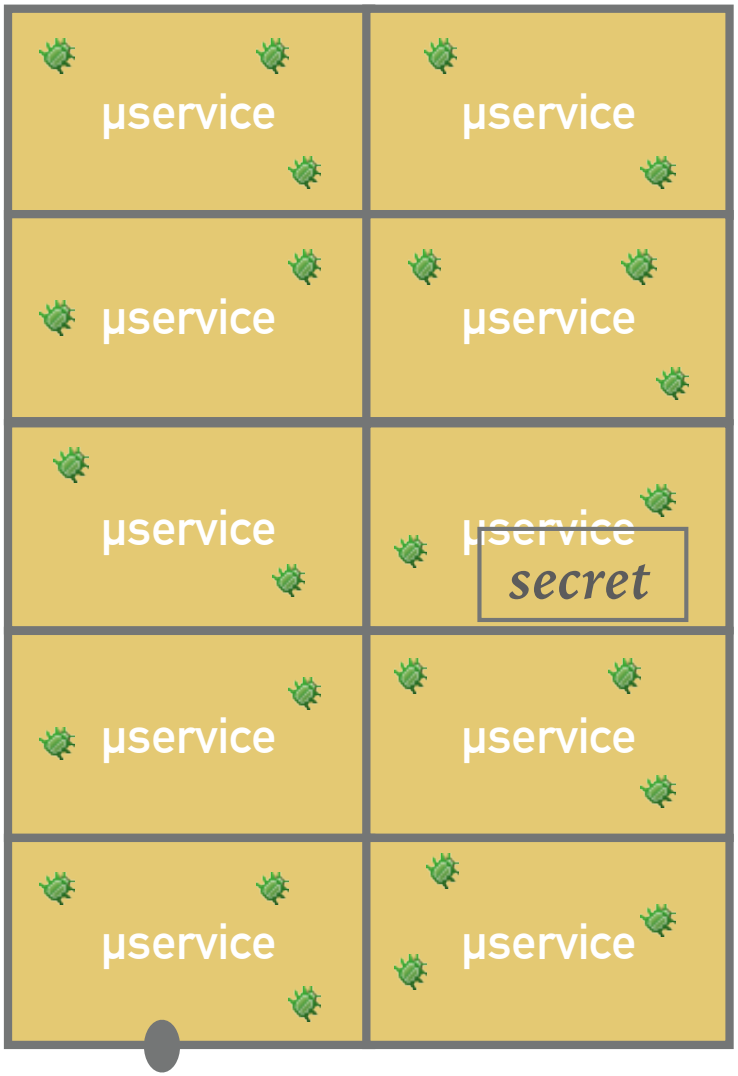
- **SGX:**
  - prevent accesses via privileged / other software
- **Smart adversary:**
  - will exploit bugs inside application code

CLOUD-NATIVE APPLICATIONS: MICROSERVICES

same address space

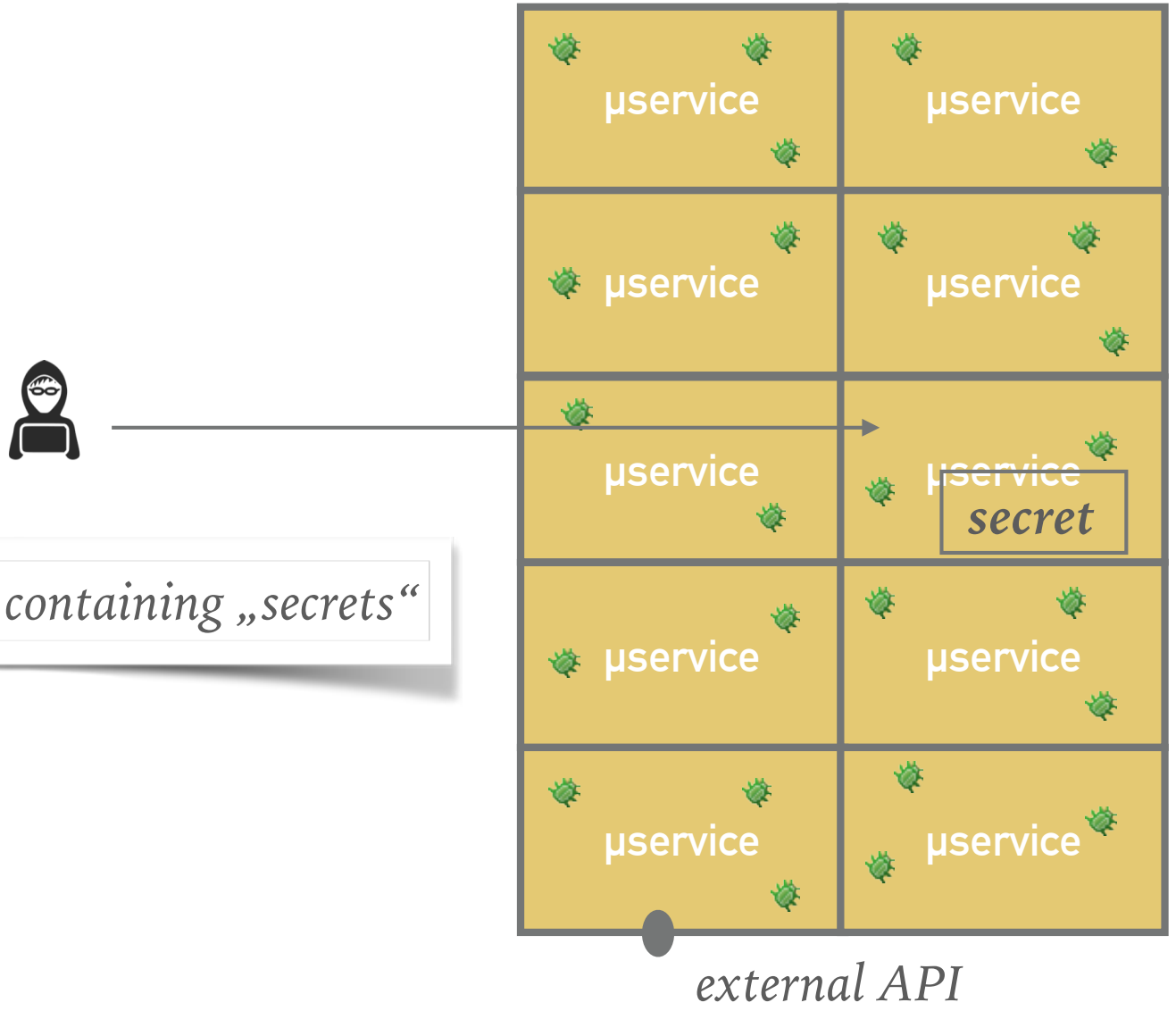


separate address spaces



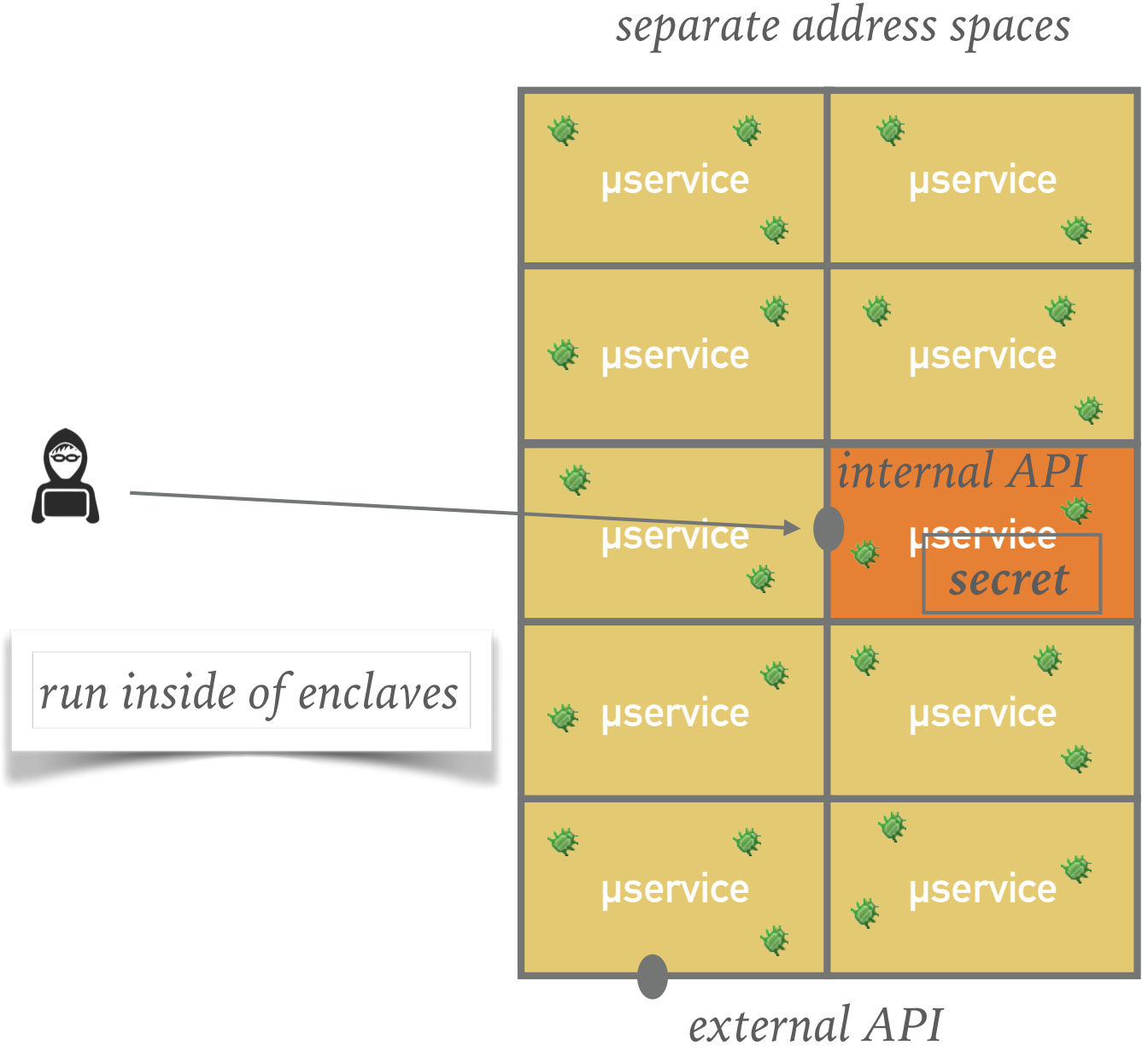
WHICH MICROSERVICES SHOULD RUN INSIDE ENCLAVES?

separate address spaces

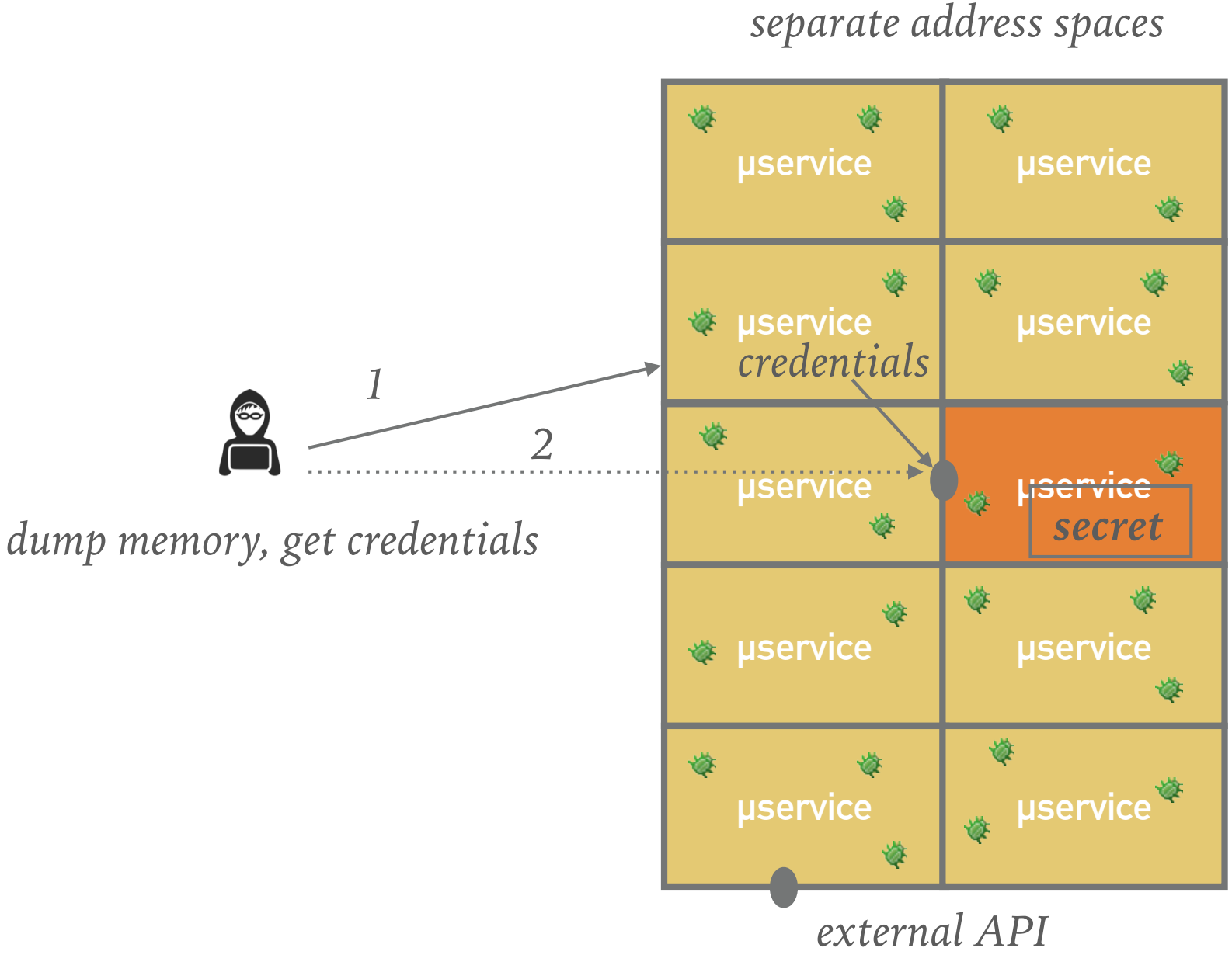


protect microservices containing „secrets“

WHICH MICROSERVICES SHOULD RUN INSIDE ENCLAVES?



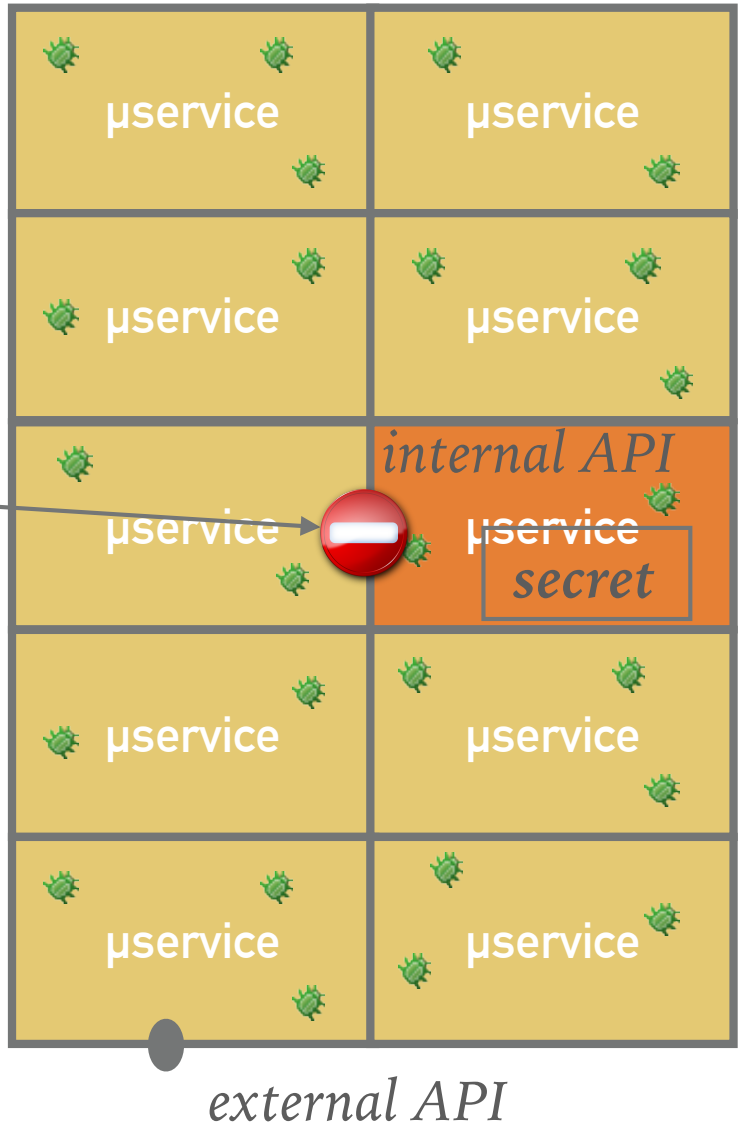
WE NEED TO PROTECT API CREDENTIALS!



APPROACH: PREVENT ACCESS TO INTERNAL APIS

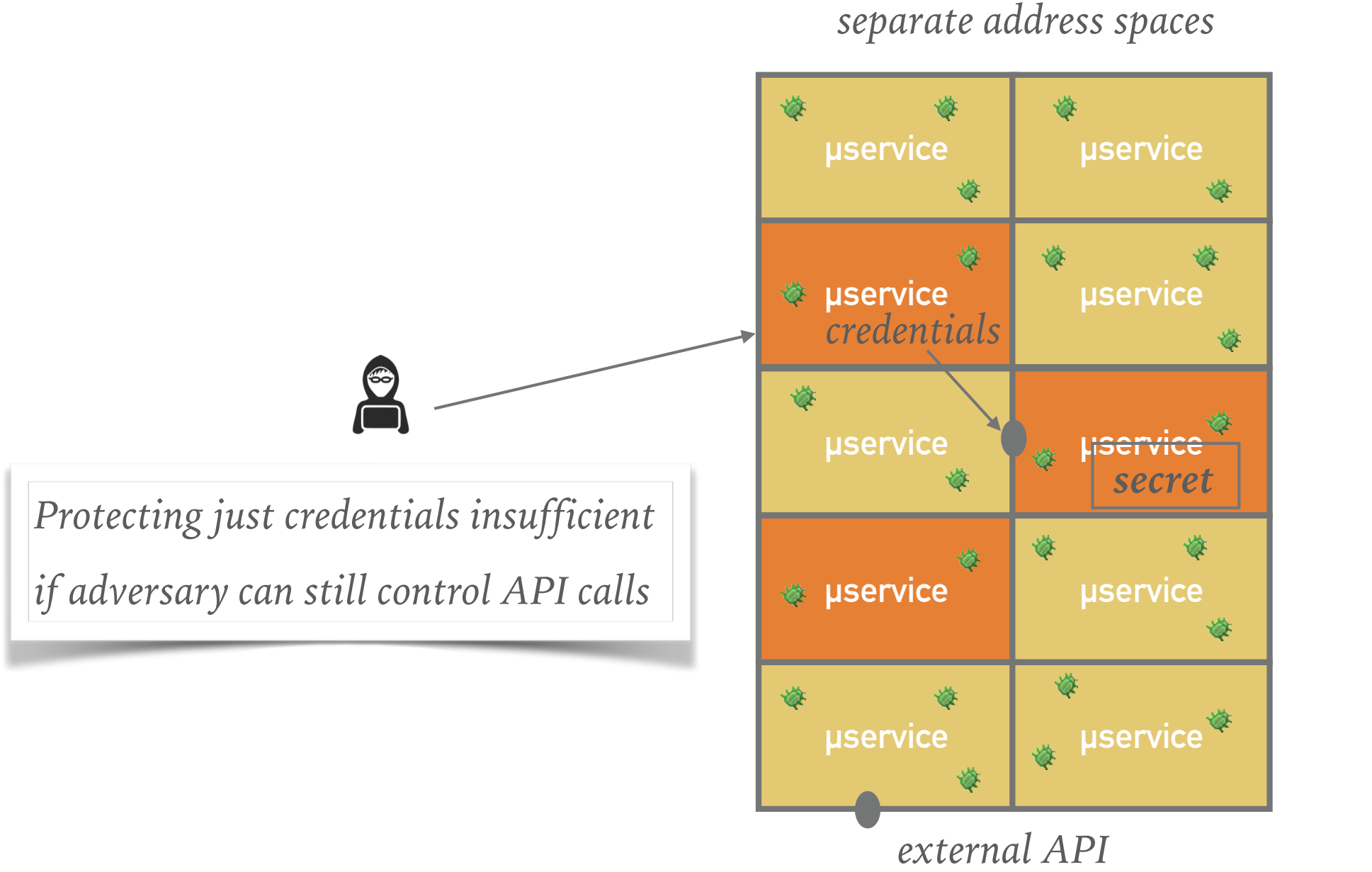
prevent adversaries from

triggering bugs inside of enclaves!



1 defect every 1700 lines

RESTRICT USAGE OF API

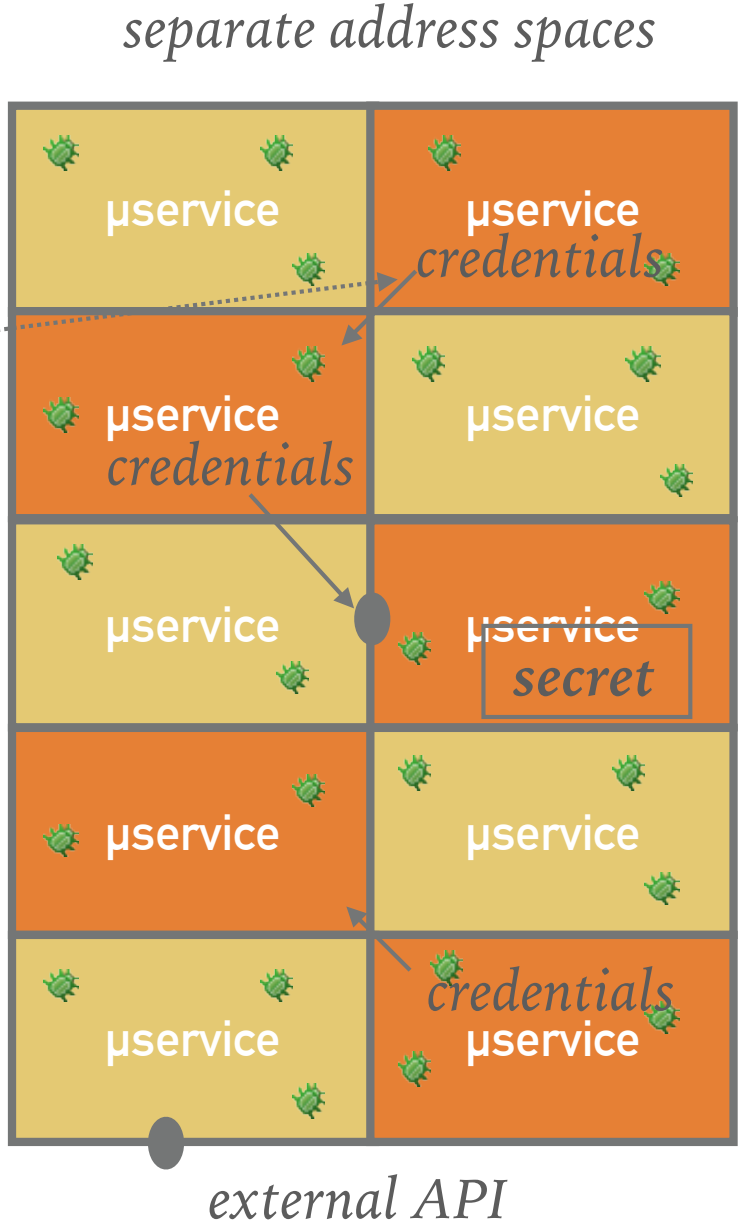


Protecting just credentials insufficient

if adversary can still control API calls

TRANSITIVE CLOSURE

Need to protect the credentials

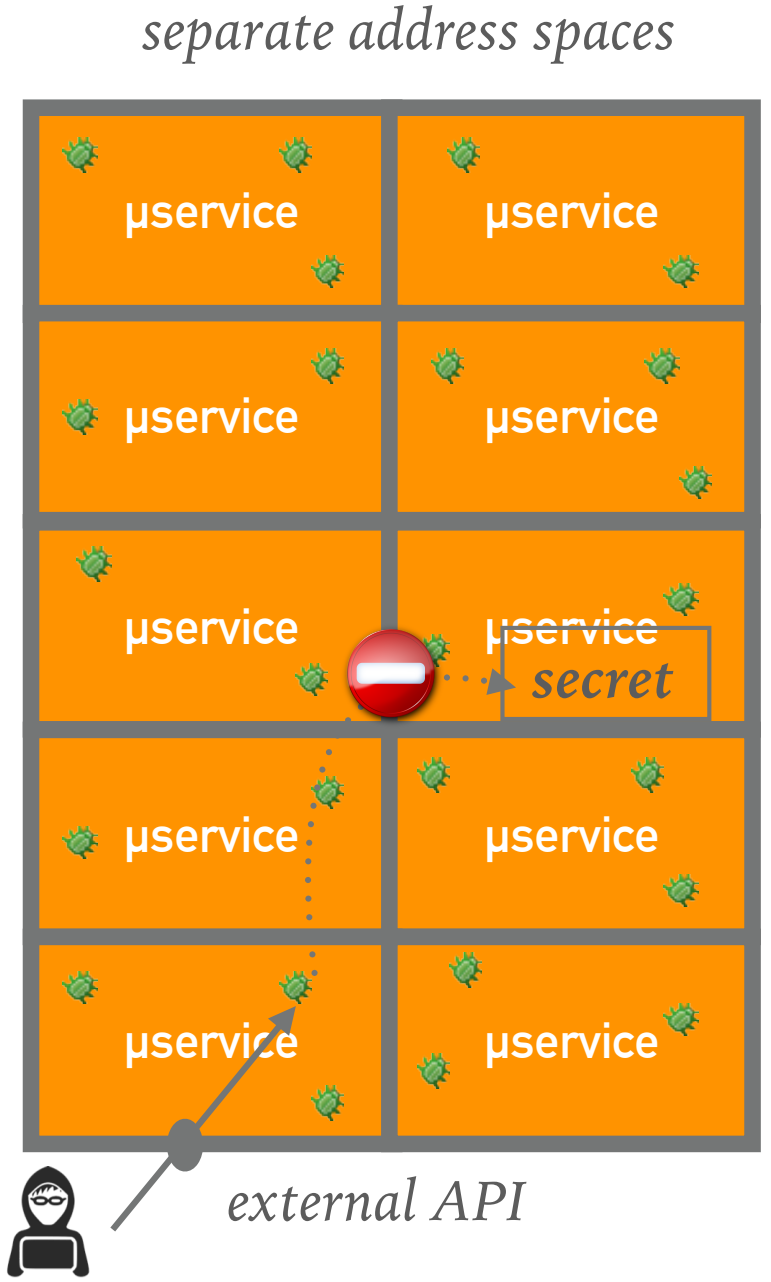


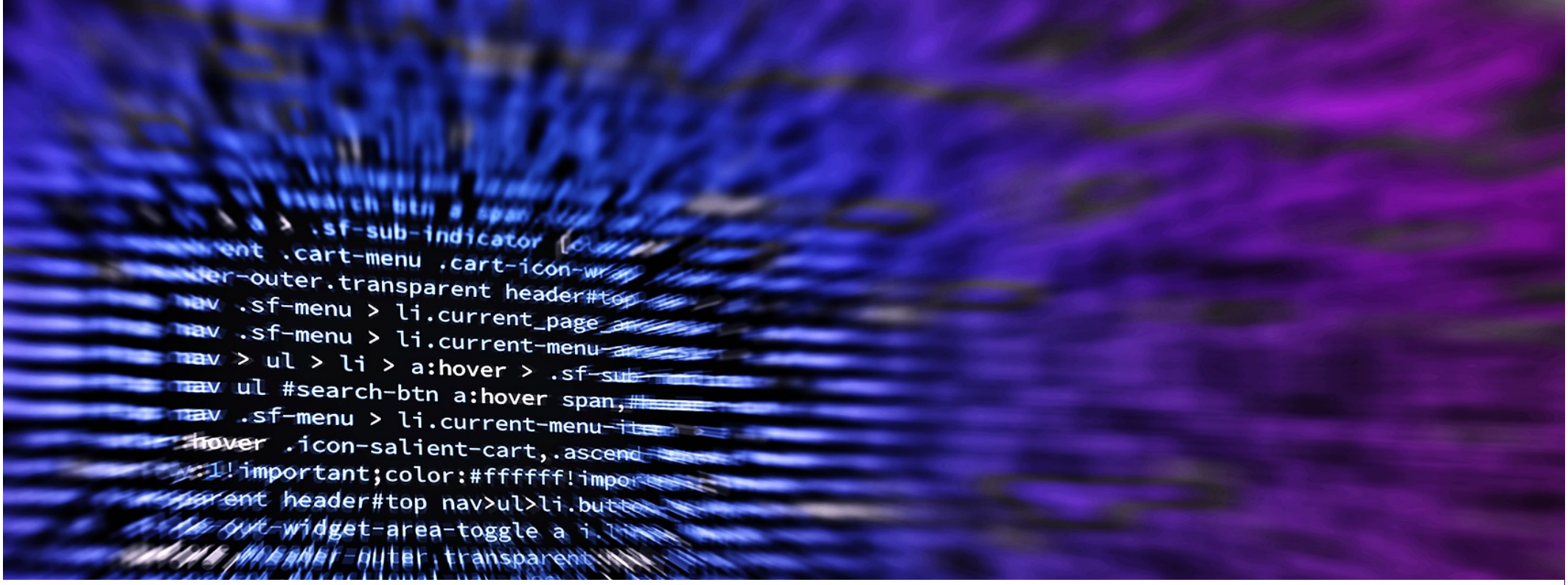
RUN ALL MICROSERVICES INSIDE ENCLAVES!

attacker must attack via

external API (or OS interface):

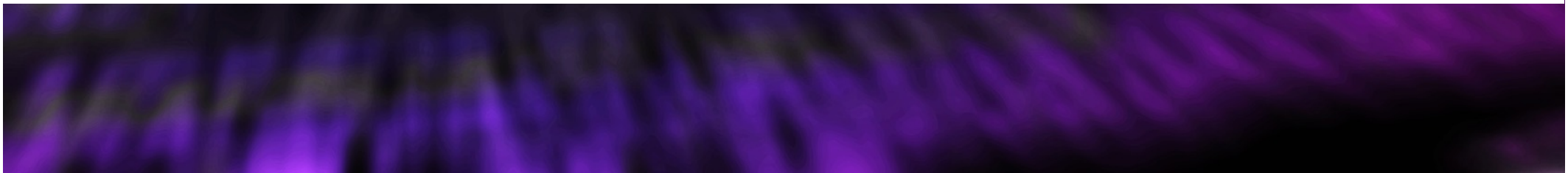
=> need to harden these APIs!



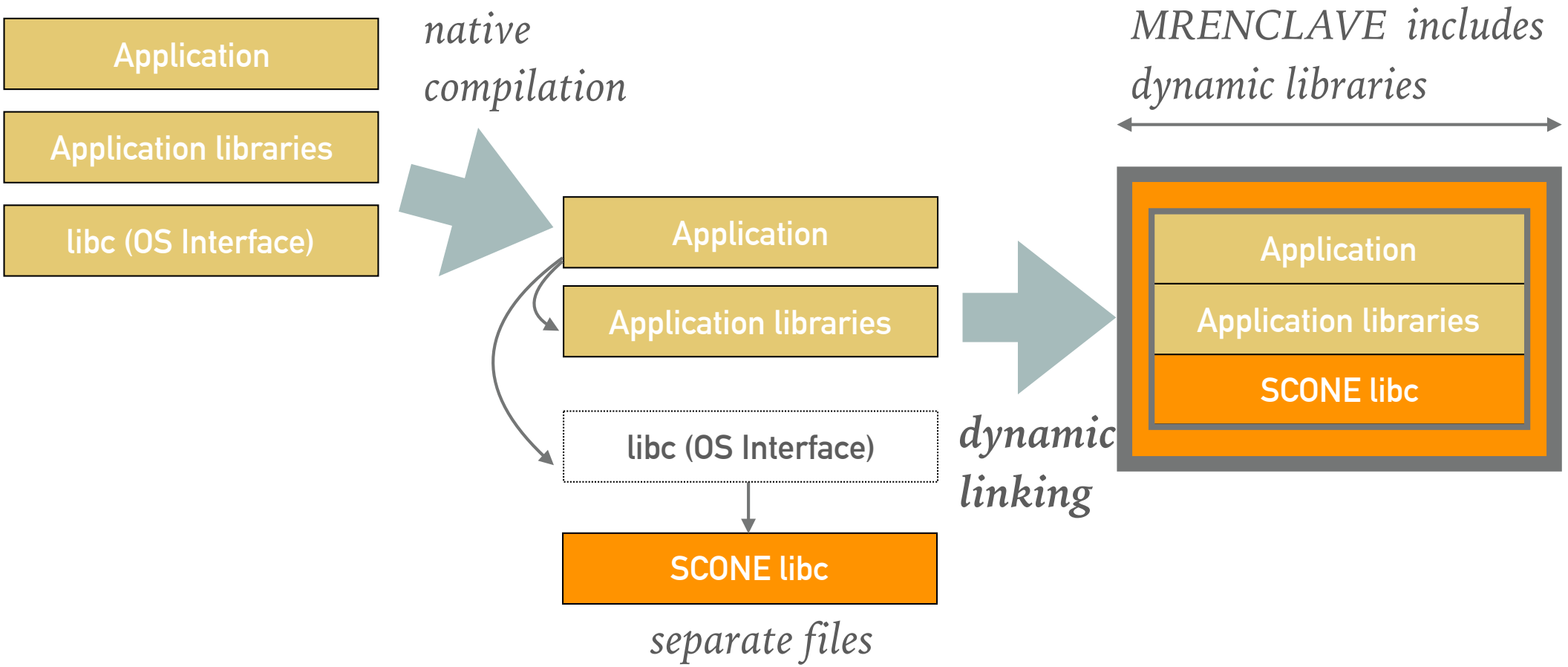


APPLICATION CODE

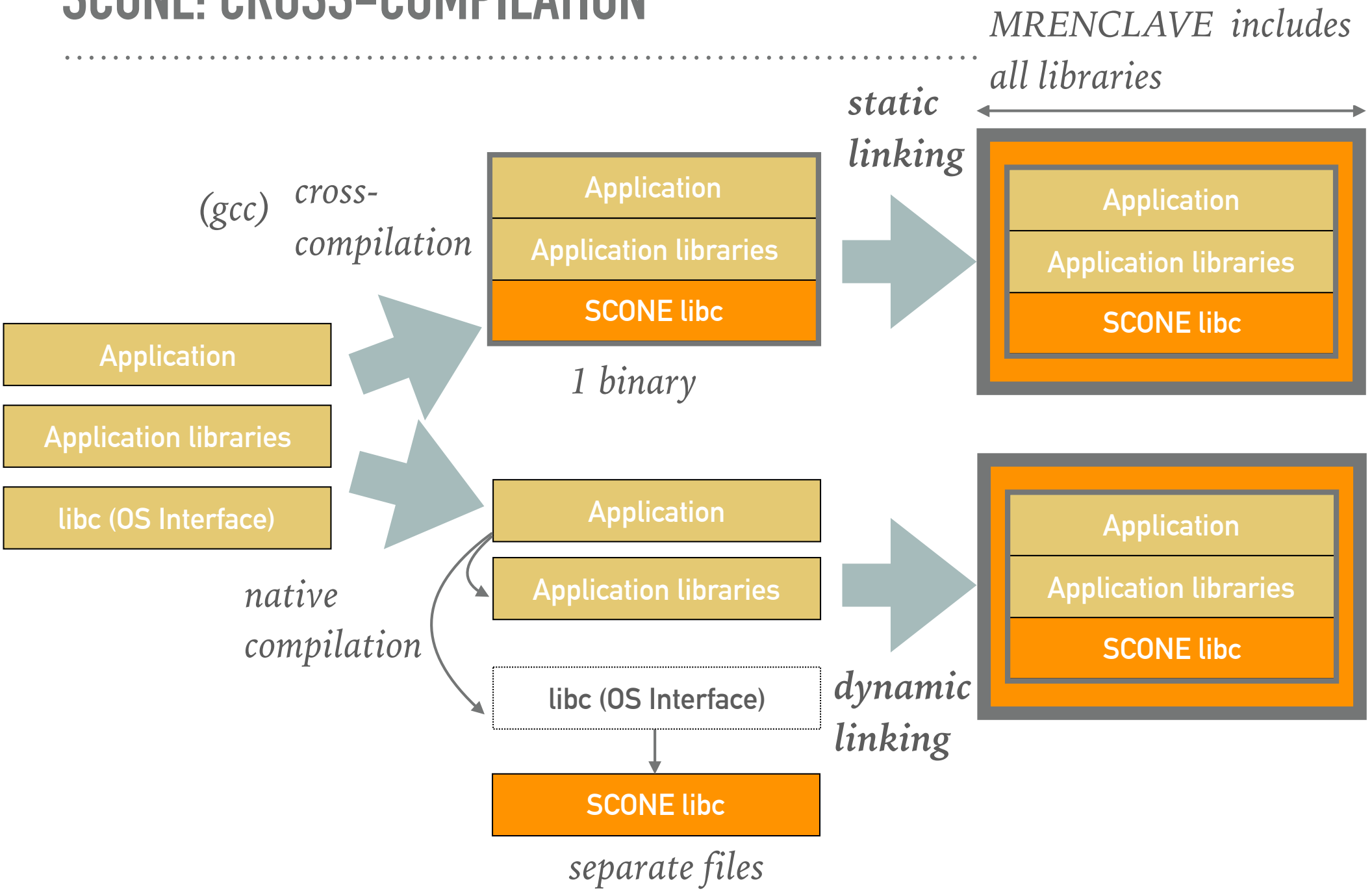
- end-2-end security without app reeingineering -



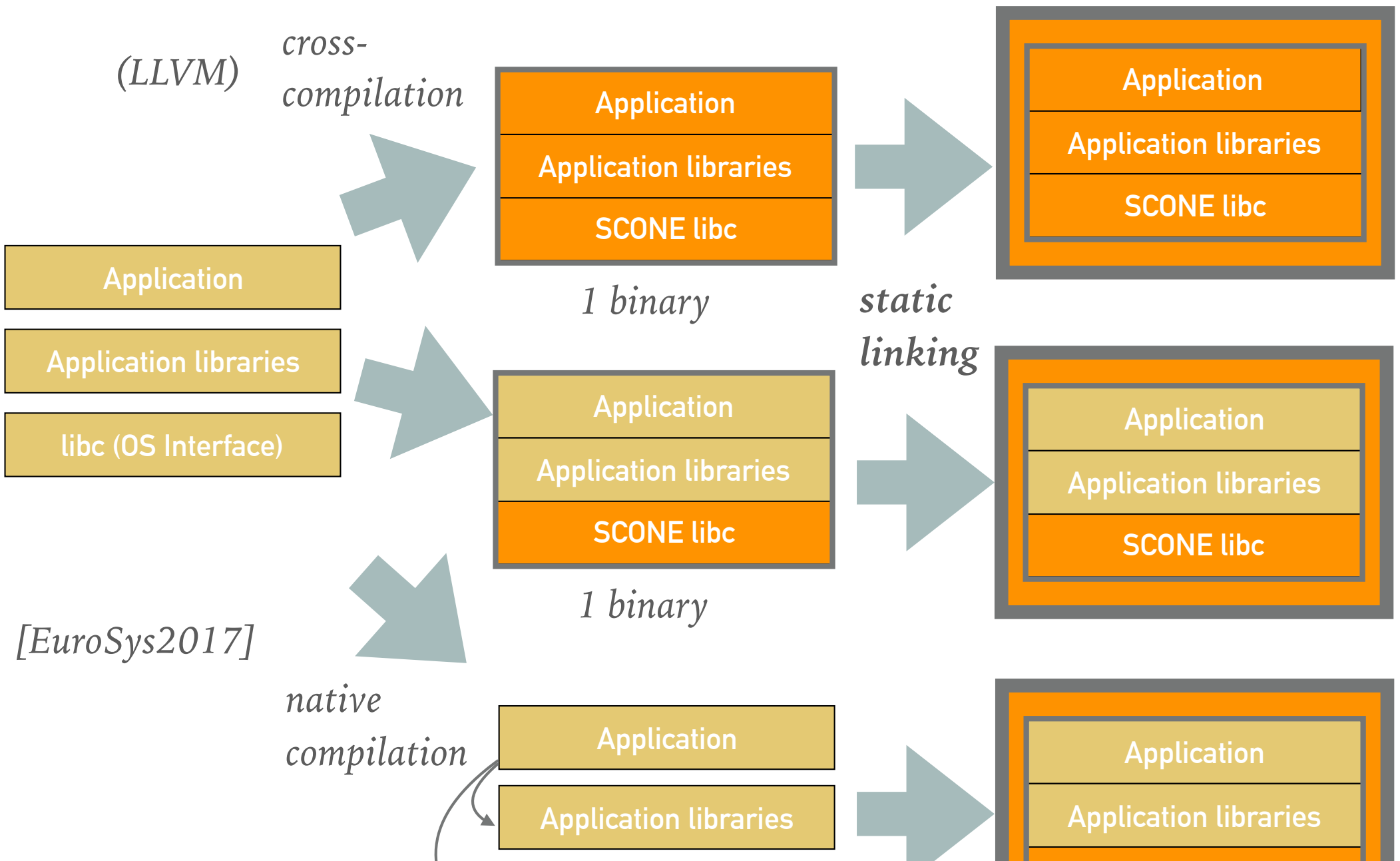
SCONE: SUPPORTS NATIVE COMPILATION

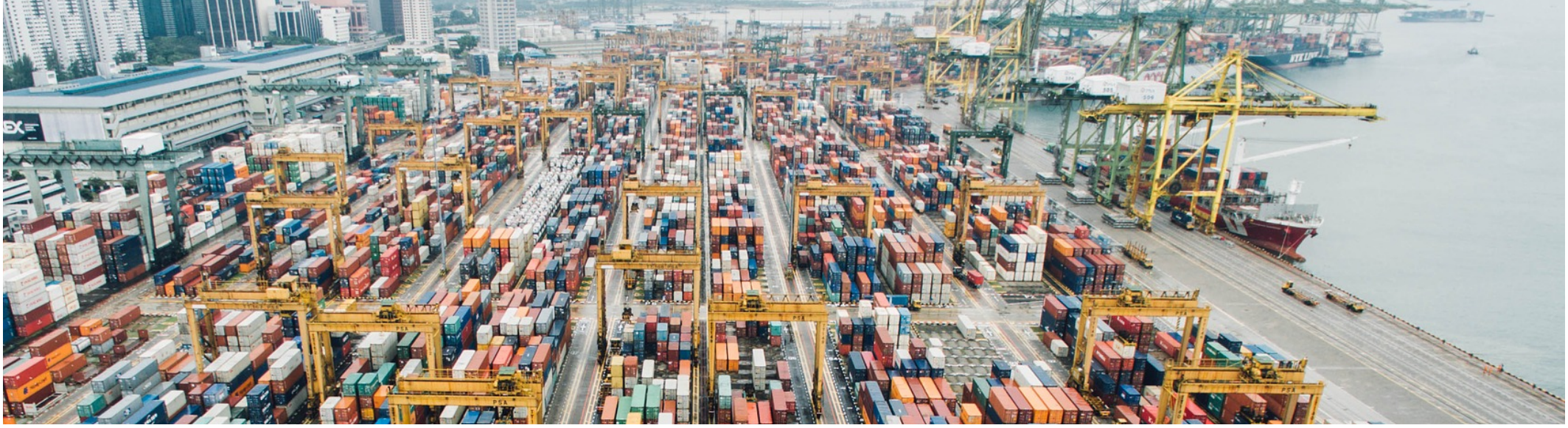


SCONE: CROSS-COMPILATION

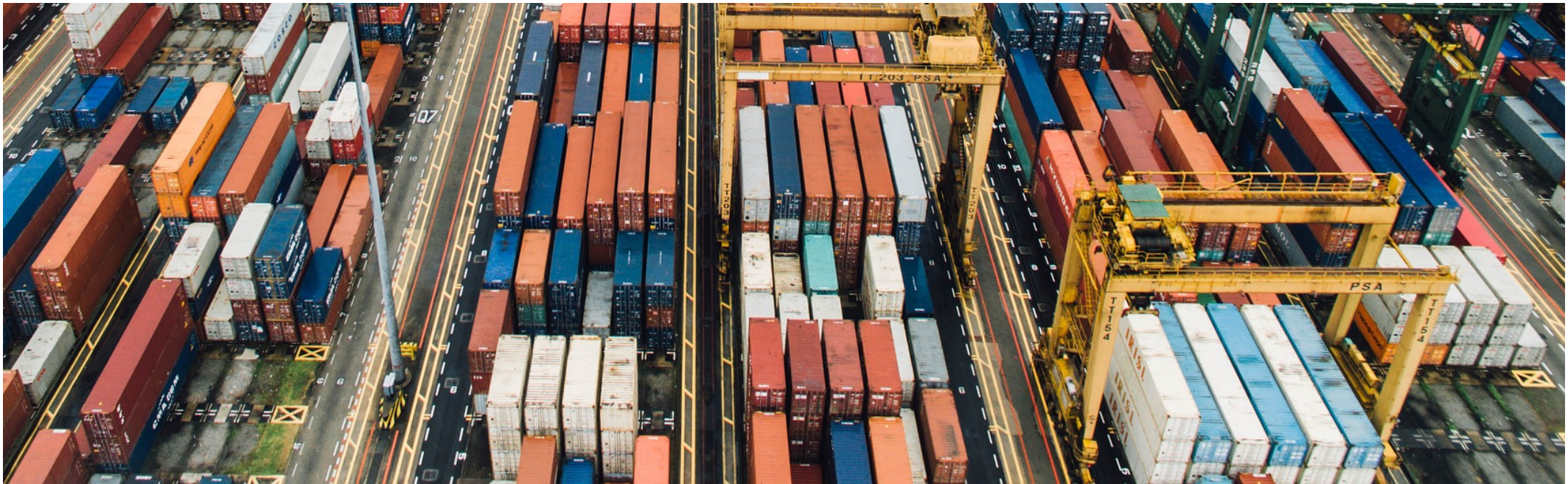


SGXBOUNDS: BOUNDS CHECKS INSIDE OF ENCLAVES

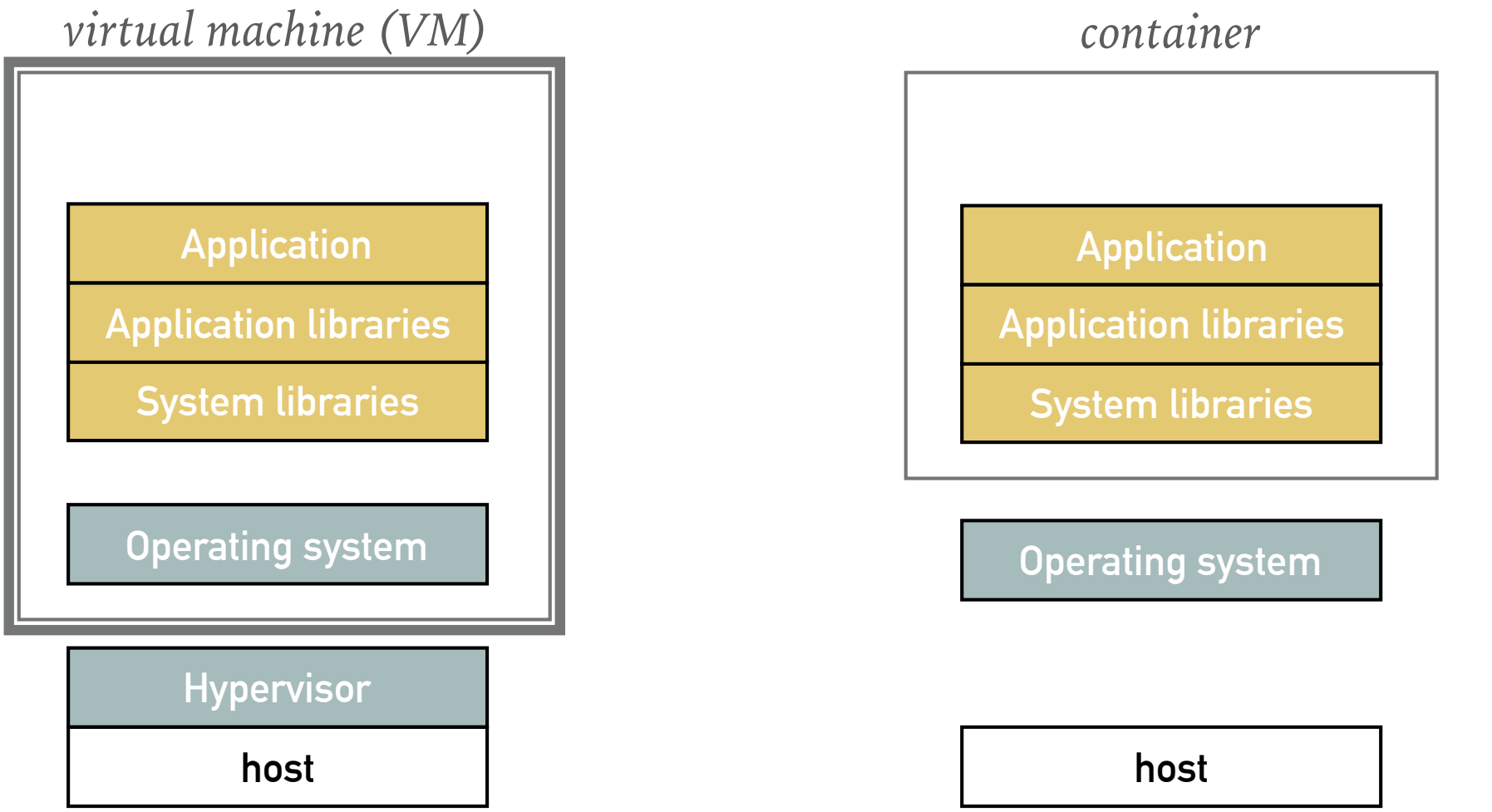




CONTAINER VS VMS



VIRTUAL MACHINES VS CONTAINER

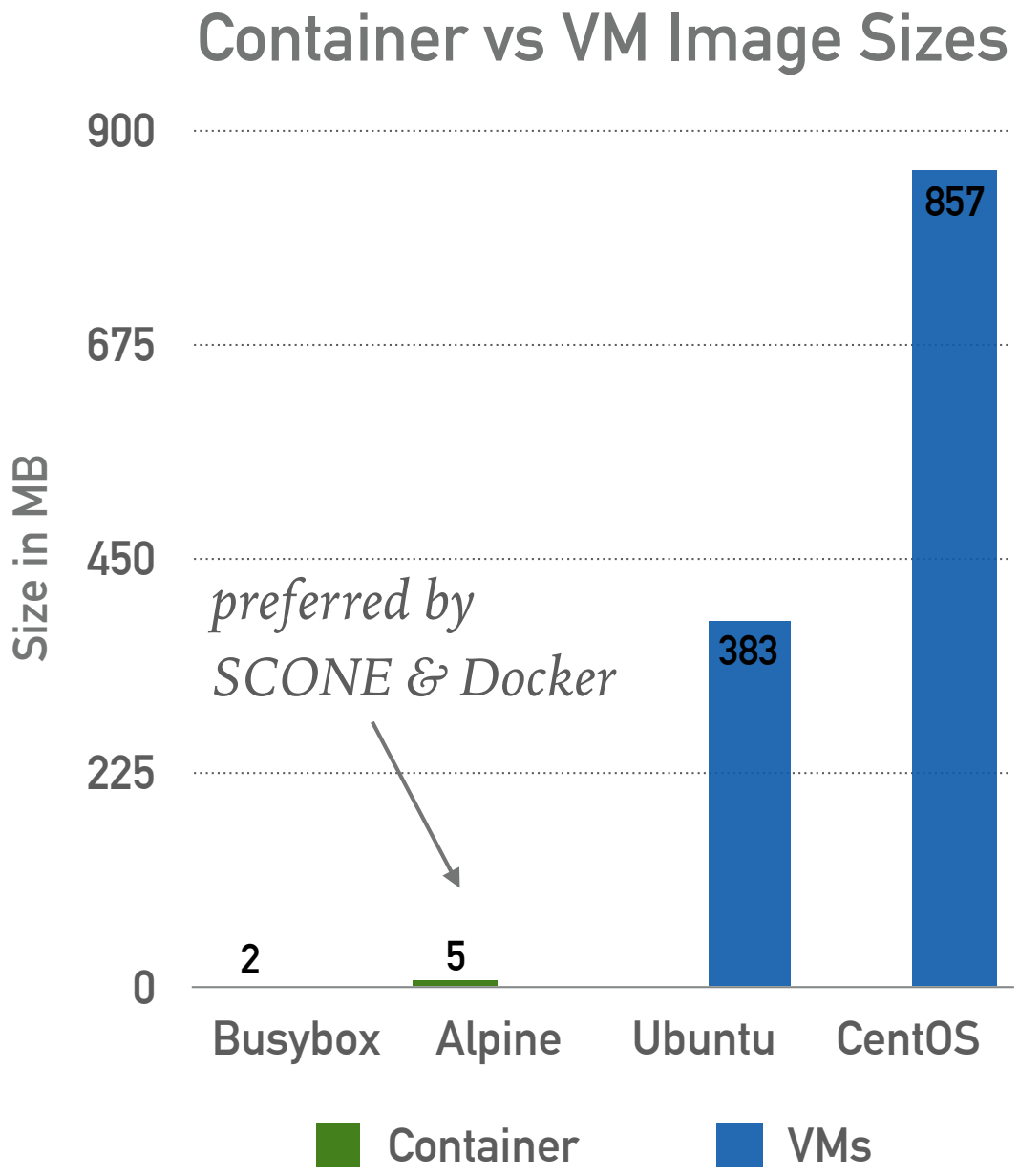


hardware virtualization

operating system virtualization

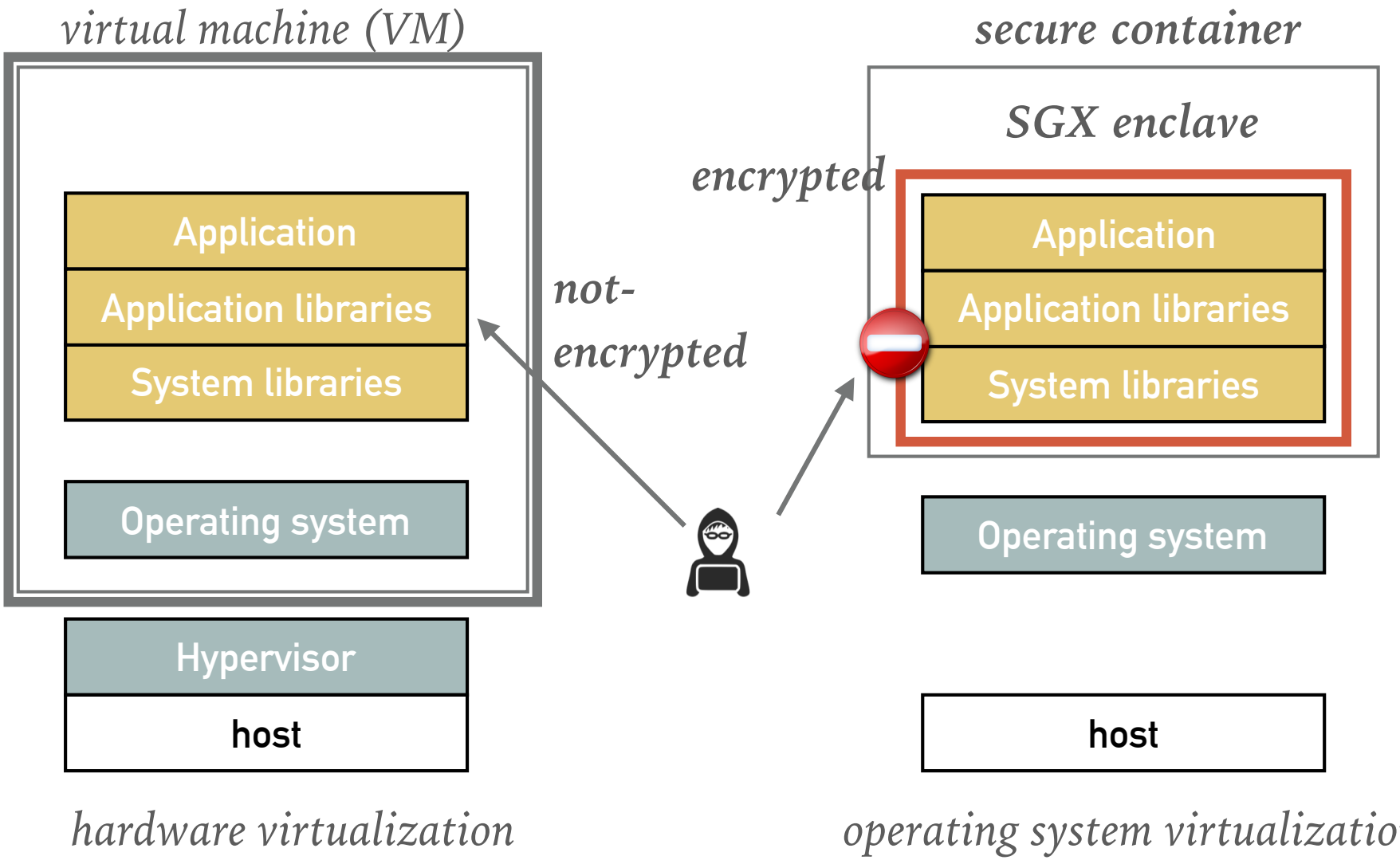
*containers are more **light-weight** but often considered **less secure***

LIGHT WEIGHT?



- **Sizes of container images:**
  - can be substantially smaller
- **Need to add application**
  - image becomes larger
- **Questions:**
  - **hardware protection?**
  - vulnerabilities in OS?
  - ease of use?

SCONE: SECURE CONTAINERS

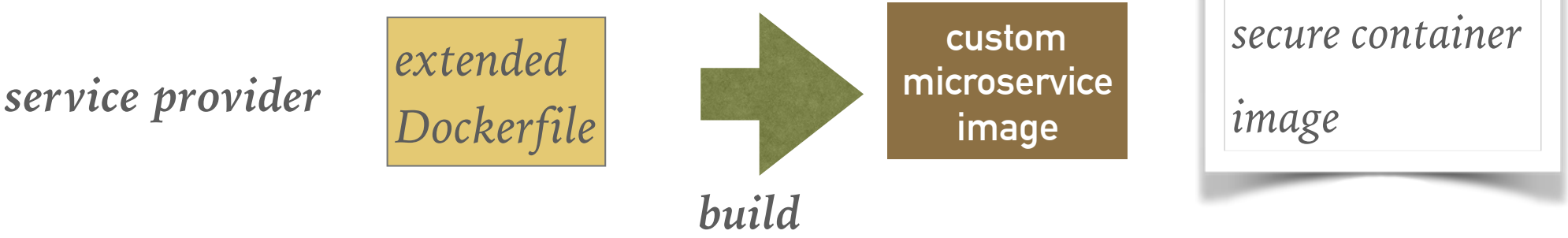


secure containers *are more light-weight and* ***more secure than VMs***

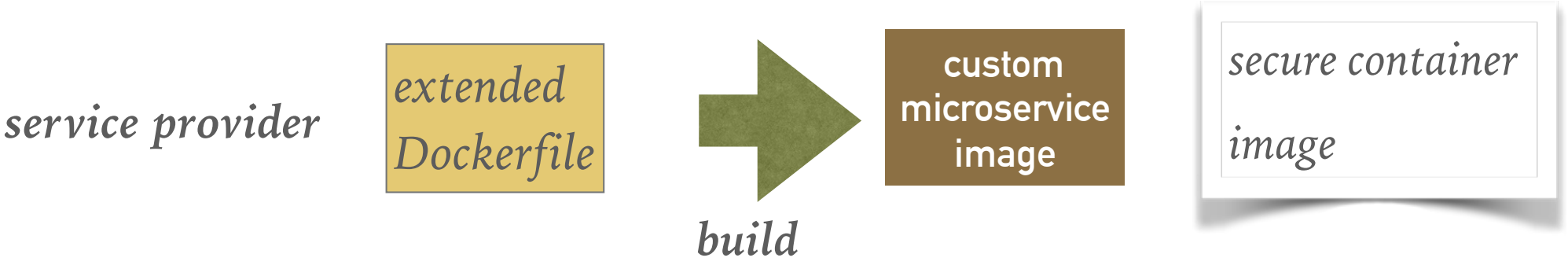
CONTAINER WORKFLOW

Ease of use!

CONTAINER WORKFLOW



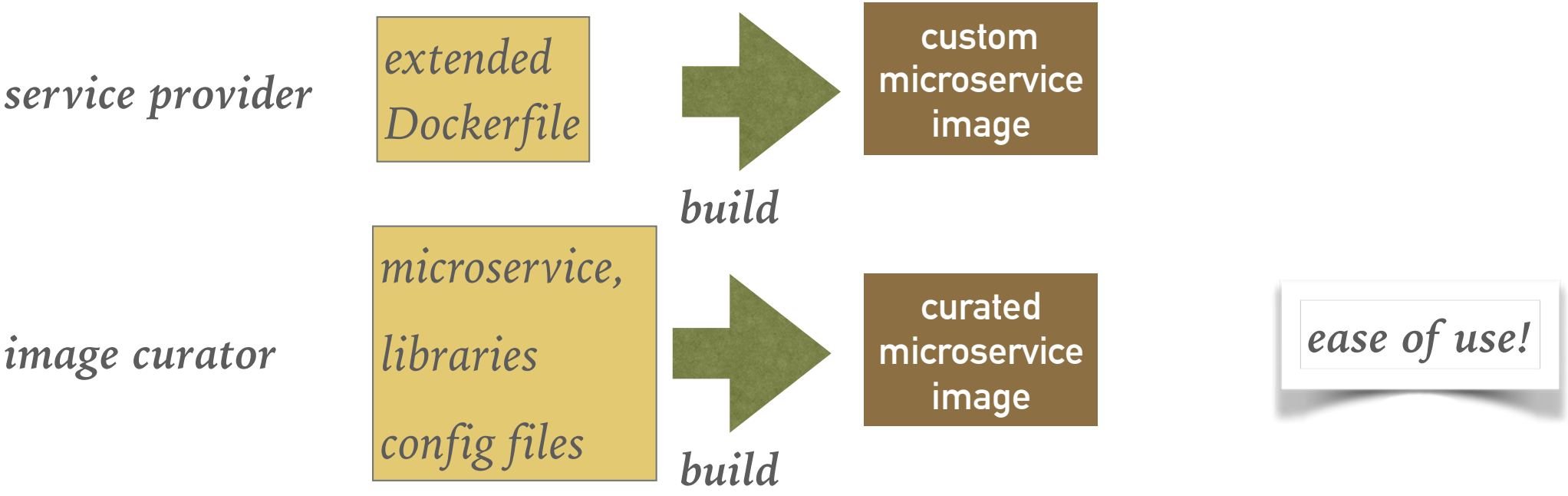
CONTAINER WORKFLOW



- **SCONE cross compilers:**
  - C, C++
  - Rust
  - GO
  - Fortran
  - Python (interpreter)
  - Java (JVM, alpha)
- **Docker**
  - to **build**, ship and deploy images

more languages soon..

CONTAINER WORKFLOW



DOCKER HUB

nginx official	5.7K STARS	10M+ PULLS	DETAILS
redis official	3.6K STARS	10M+ PULLS	DETAILS
mysql official	4.1K STARS	10M+ PULLS	DETAILS
mongo official	3.1K STARS	10M+ PULLS	DETAILS

...

hub.docker.com/explore

SCONE CURATED IMAGES (WORK IN PROGRESS)

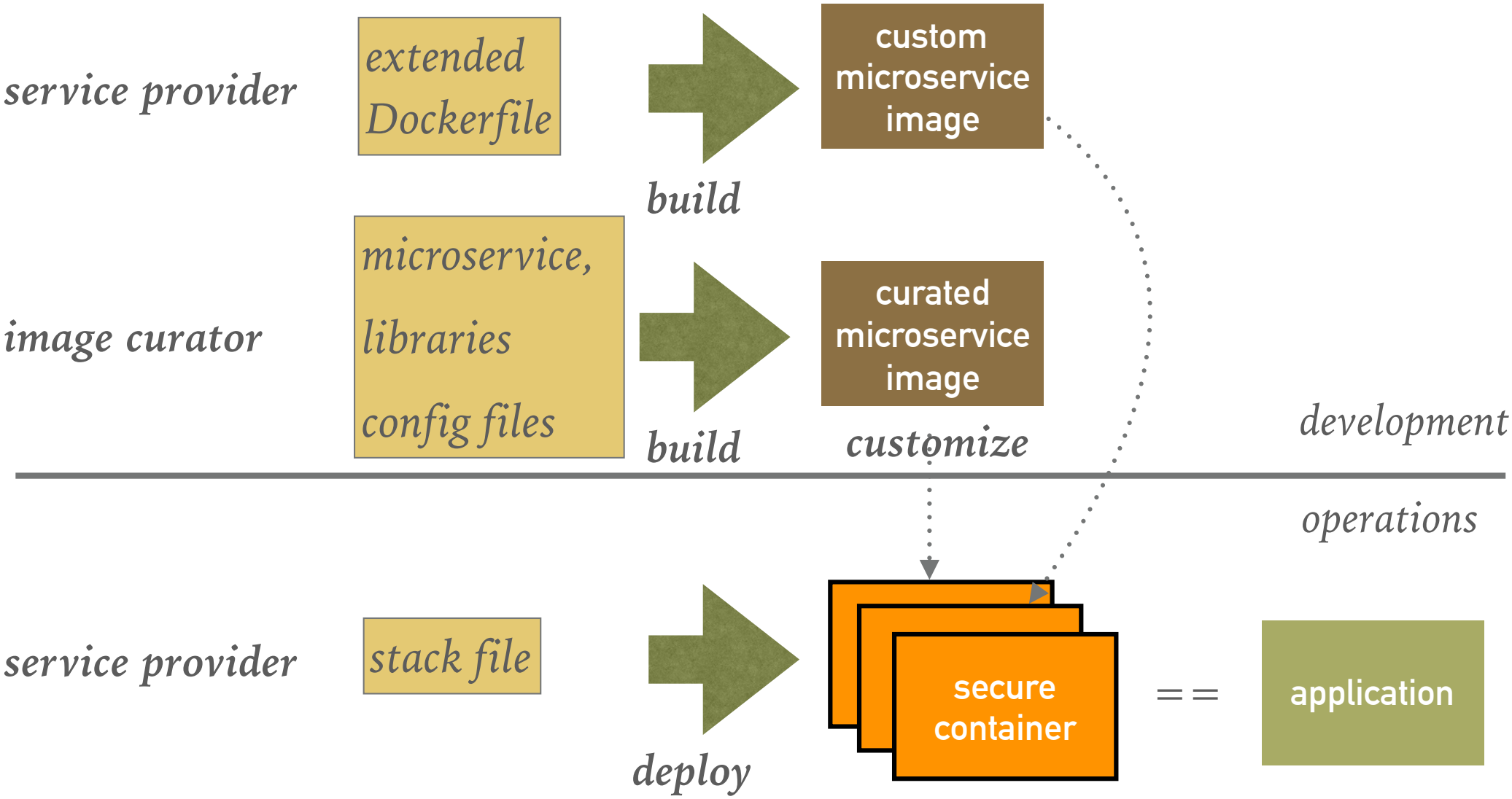
nginx official	→	*nginx SCONE image*	5.7K STARS	10M+ PULLS	DETAILS
redis official	→	*redis SCONE image*	3.6K STARS	10M+ PULLS	DETAILS
mysql official	→	*mysql SCONE image*	4.1K STARS	10M+ PULLS	DETAILS
mongo official	→	*mongo SCONE image*	3.1K STARS	10M+ PULLS	DETAILS

...

SCONE images are shielded and tuned for SGX

hub.docker.com/explore

CONTAINER WORKFLOW



COMPOSE EXAMPLE

HOW TO DISTRIBUTE SECRETS?

```
mysql-master:
  environment:
   MYSQL_ROOT_PASSWORD: rootpass
   MYSQL_DATABASE: messenger
   MYSQL_USER: messenger
   MYSQL_PASSWORD: messenger
  tty: true
  tty-key: mysecret
  image: mysql
   MRENCLAVE: 0x3394940494
   FSPFKEY:  topsecret
  stdin_open: true
```

- State of the art:
  - put passwords in compose file
- Problem:
  - Docker engine is not trusted

Bad practice to put secrets in compose file!

HOW TO DISTRIBUTE SECRETS?

```
mysql-master:
  environment:
   MYSQL_ROOT_PASSWORD: $rootpass
   MYSQL_DATABASE: messenger
   MYSQL_USER: messenger
   MYSQL_PASSWORD: $messenger
  tty: true
  tty-key: $messenger
  image: mysql
   MRENCLAVE: 0x3394940494
   FSPFKEY:  $fspfkey
  stdin_open: true
```

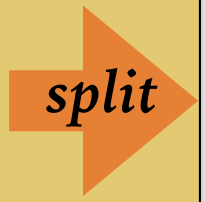
Use variables instead

- **Problem**: team member leaves
  - We would need to rekey
- **We support:**
  - variables - value retrieved from a keystore

EXAMPLE: MYSQL

```
mysql-master:
  environment:
    MYSQL_ROOT_PASSWORD: $rootpass
    MYSQL_DATABASE: messenger
    MYSQL_USER: messenger
    MYSQL_PASSWORD: $messenger
  tty: true
  tty-key: $messenger
  image: mysql
    MRENCLAVE: 0x3394940494
    FSPFKEY:  $fspfkey
  stdin_open: true
```

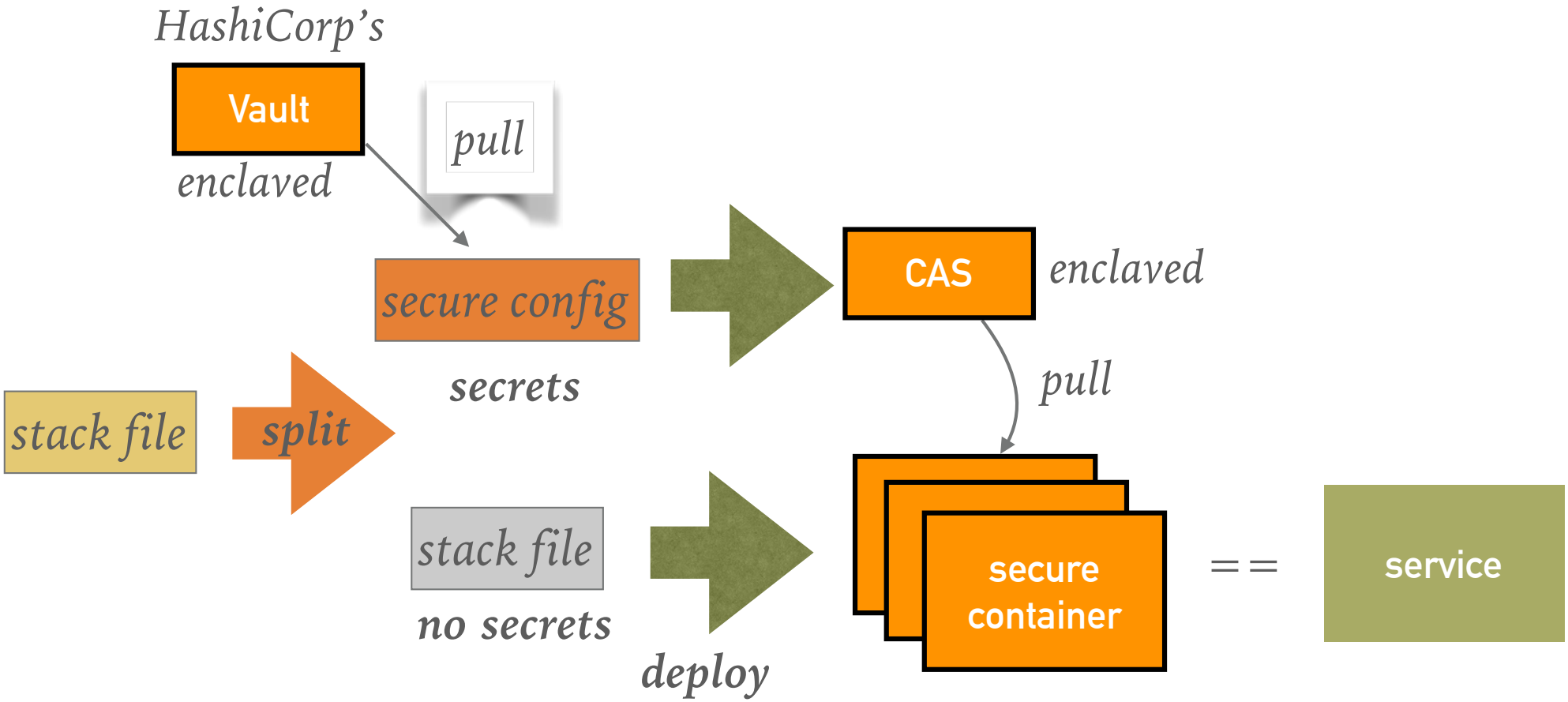
extended compose file




```
mysql-master:                    private stack file
environment:
  MYSQL_ROOT_PASSWORD: rootpass
  MYSQL_DATABASE: messenger
  MYSQL_USER: messenger
  MYSQL_PASSWORD: messenger
tty-key: mysecret
MRENCLAVE: 0x3394940494
FSPFKEY: topsecret                          CAS
```

```
mysql-master:                    standard stack file
  environment:
    APPID:  012345
  tty: true
  image: mysql
  stdin_open: true                          Docker
```

APPROACH: RETRIEVE SECRETS FROM VAULT



SCONE SUMMARY

- **Simplifies moving application to SGX enclaves**
- **Provides:**
  - Secure application configuration
  - Transparent attestation
  - Secure main memory
  - Integration with secure key store
  - Transparent file protection
  - Transparent TCP encryption
  - Ease of use (via Docker integration)

QUESTIONS?

https://sconedocs.github.io/ *http://scontain.com*