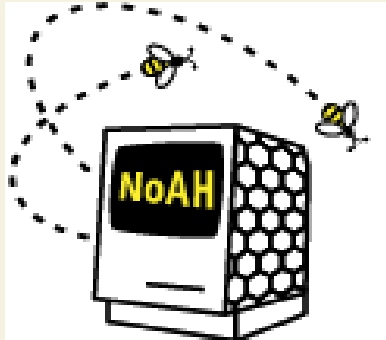


The NoAH approach to zero-day worm detection



Asia Slowinska
Vrije Universiteit, Amsterdam

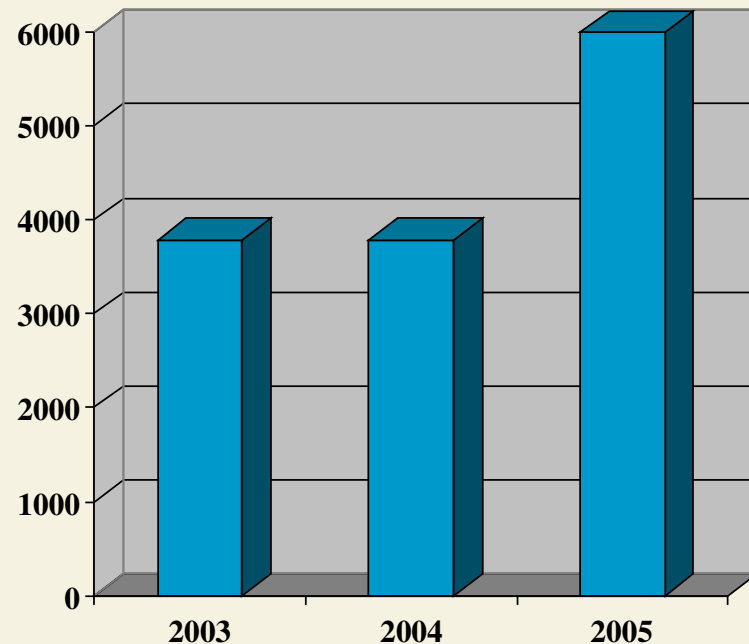


What is NoAH?

- ~ NoAH is a Specific Support Action in the Sixth Framework Programme of the European Union
- ~ Start: 1st April 2005
- ~ End: 31st March 2008
- ~ Homepage: <http://www.fp6-noah.org/>

Why?

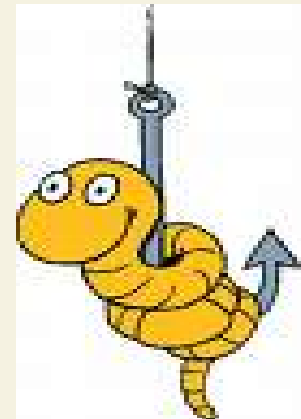
- ~ Too many vulnerabilities
- ~ New worm attacks
- ~ Human intervention too slow
- ~ Current solutions are not perfect
 - ~ Time consuming
 - ~ Inaccurate



CERT/CC Reported Vulnerabilities

Why?

- ~ Too many vulnerabilities
- ~ New worm attacks
- ~ Human intervention too slow
- ~ Current solutions are not perfect
 - ~ Time consuming
 - ~ Inaccurate



Why?

- ~ Too many vulnerabilities
- ~ New worm attacks
- ~ Human intervention too slow
- ~ Current solutions are not perfect
 - ~ Time consuming
 - ~ Inaccurate



Why?

- ~ Too many vulnerabilities
- ~ New worm attacks
- ~ Human intervention too slow
- ~ Current solutions are not perfect
 - ~ Time consuming
 - ~ Inaccurate



Goals

- ~ Design and develop infrastructure for security monitoring based on honeypots technology
- ~ Detect most common attack vectors
 - ~ Detect worms in early stage of spreading
- ~ Gather information about attacks
- ~ Generate signatures

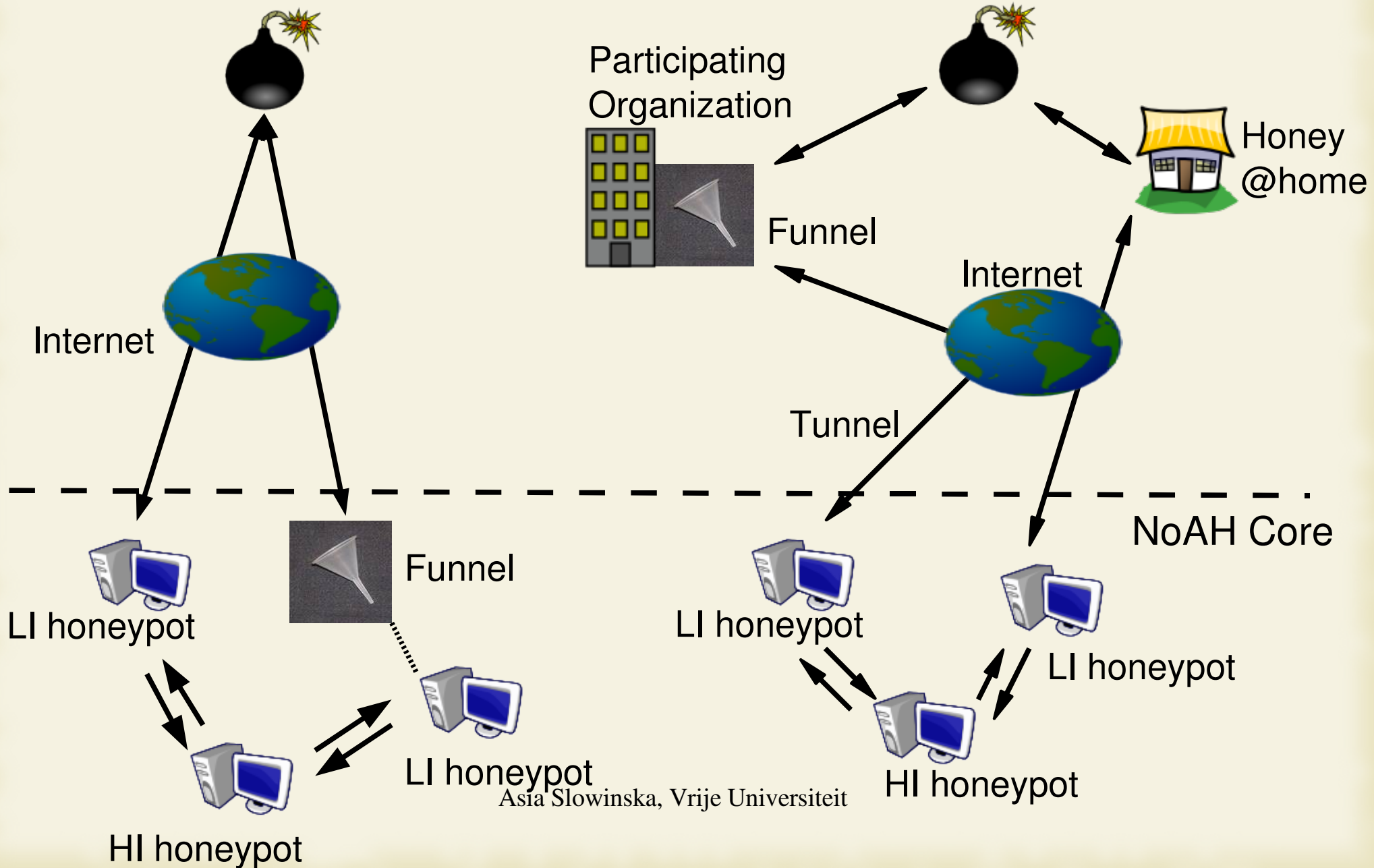


Honeypots

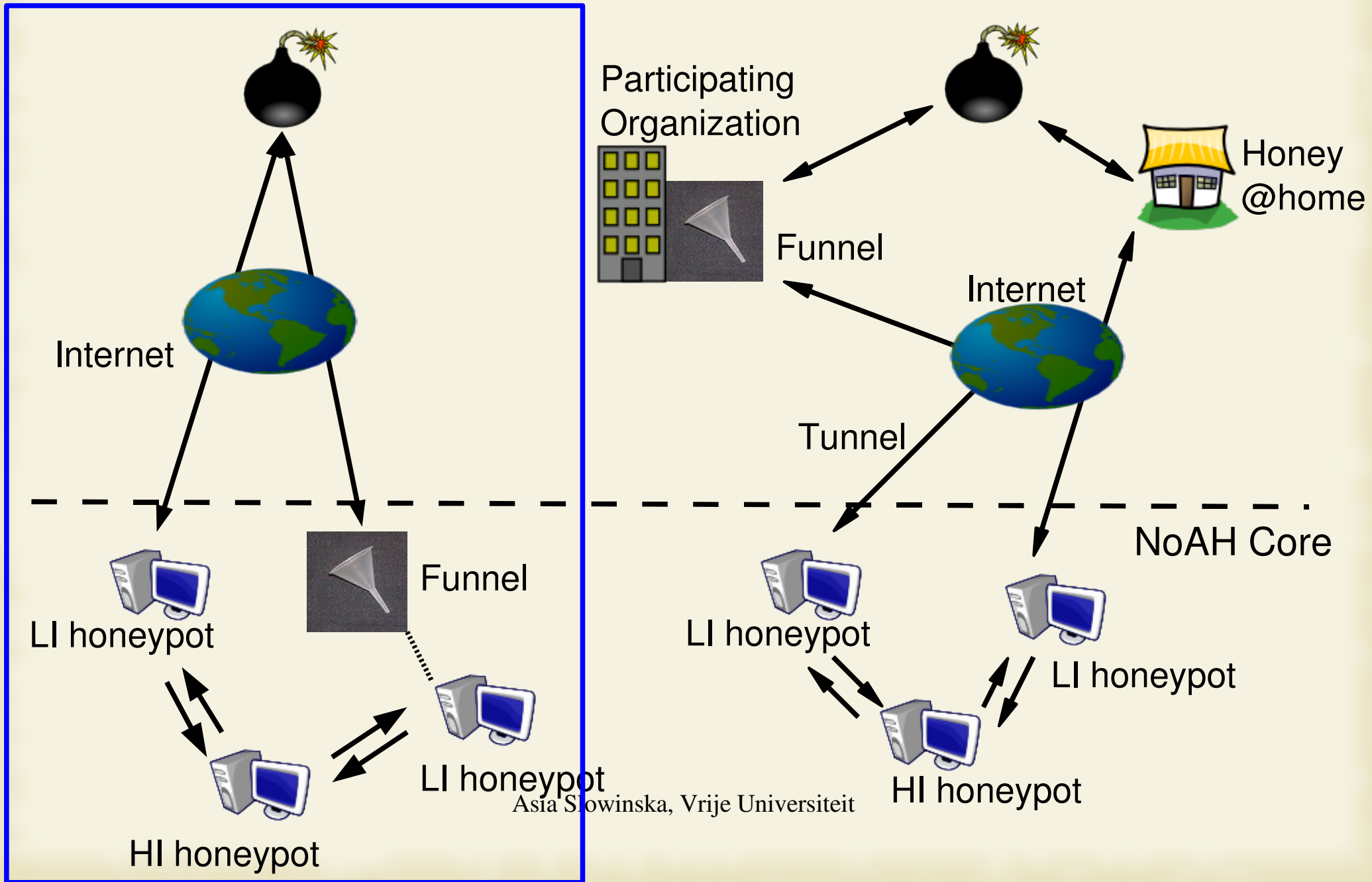


- ~ A computer system acting as a decoy
 - ~ does not provide regular services
 - ~ lures in potential hackers in order to study their activities
 - ~ honeypots in NoAH listen to unused IP address space, called further *dark space*
- ~ Two basic types of honeypots
 - ~ low interaction (LI) – emulate services
 - ~ high interaction (HI) – run real applications

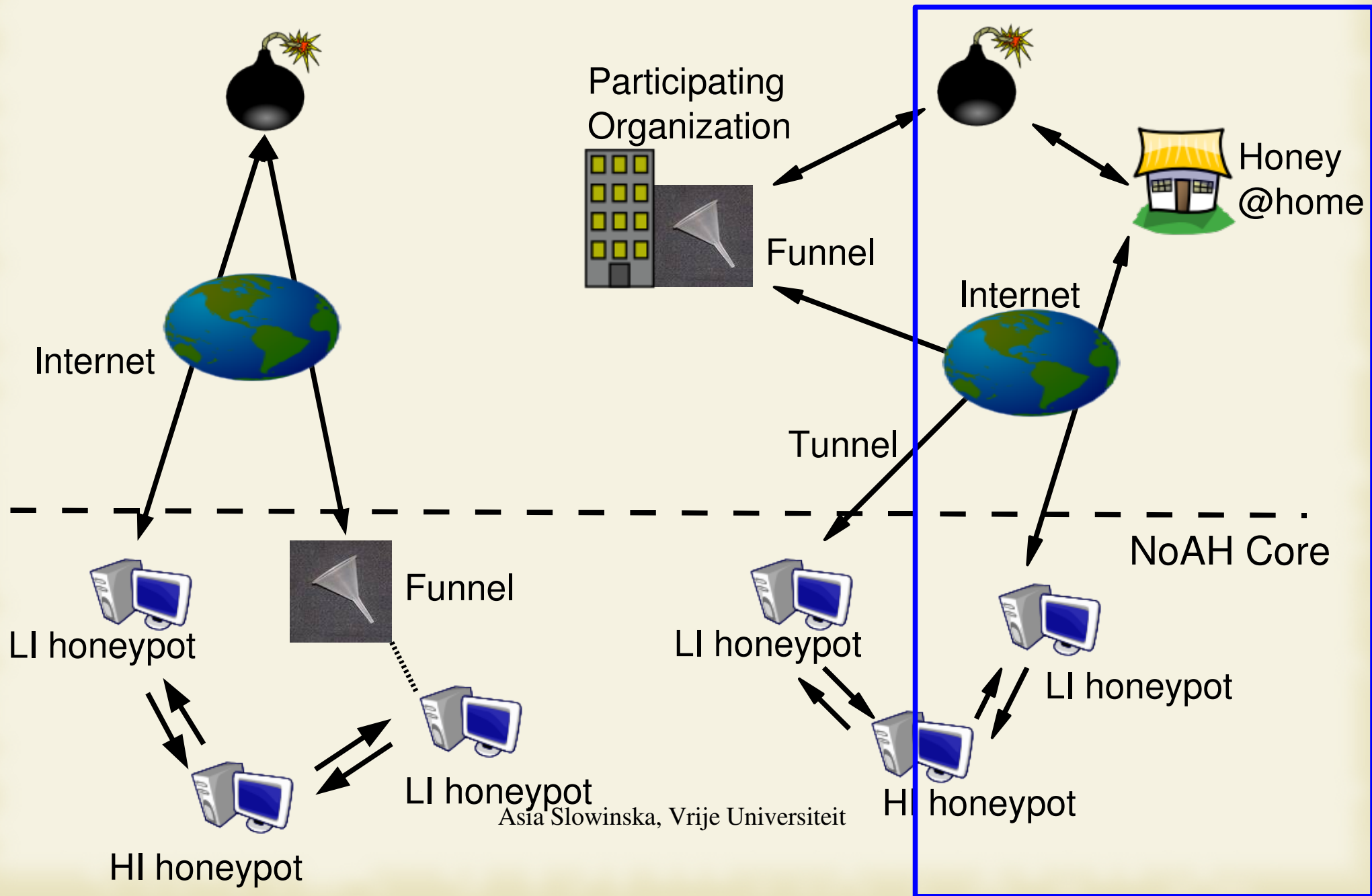
NoAH architecture



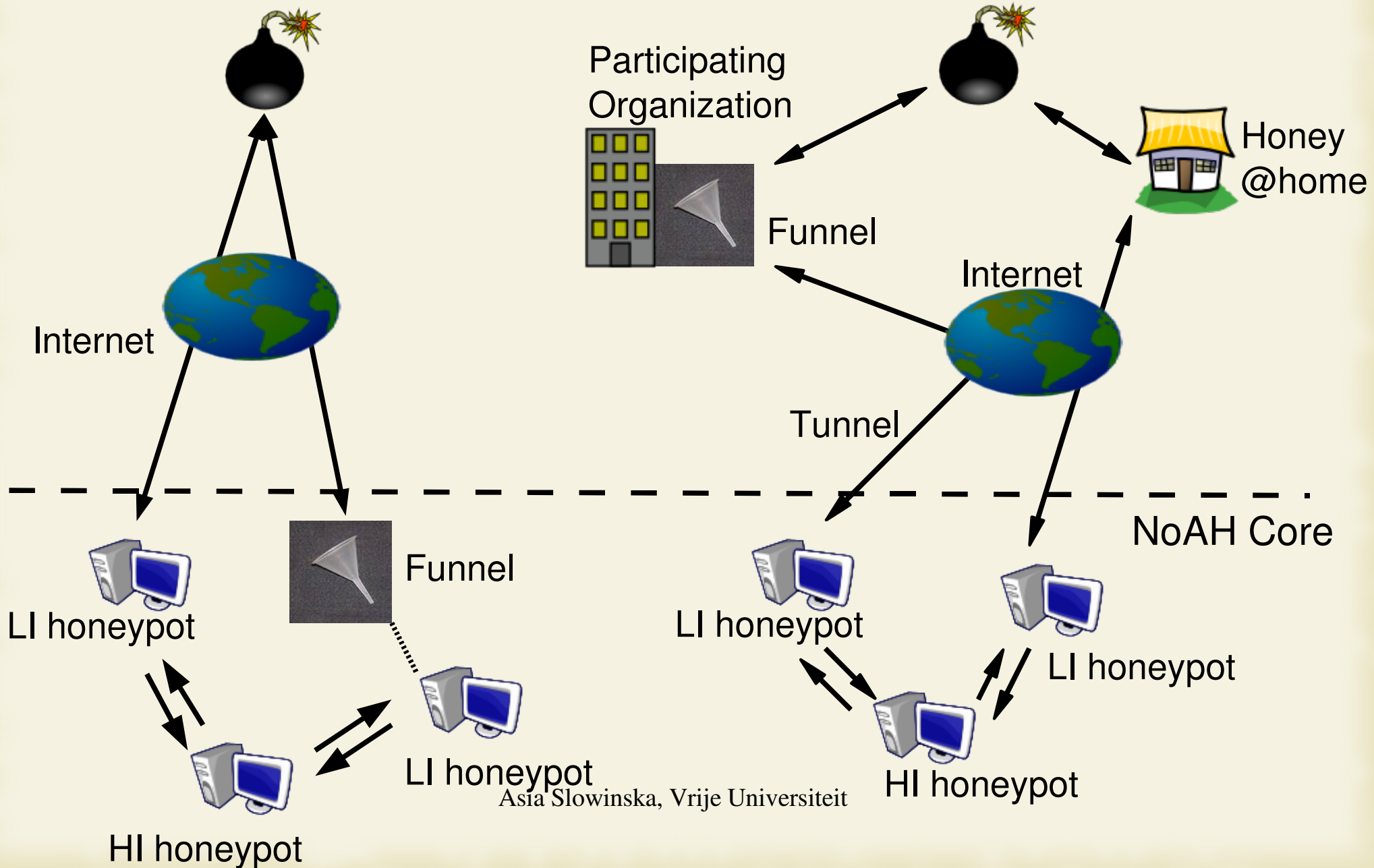
NoAH architecture



NoAH architecture



NoAH architecture

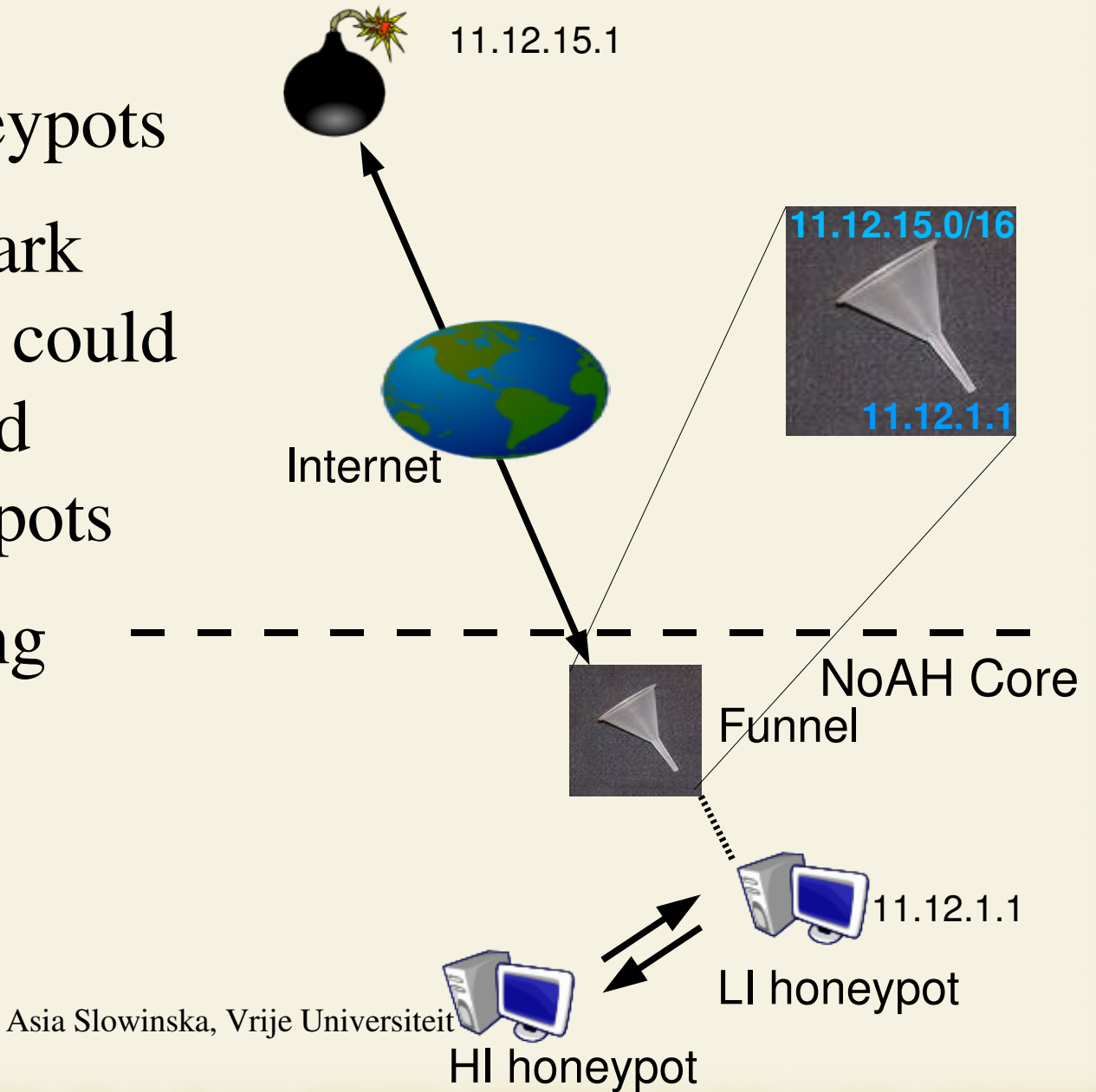


Core organizations

- ~ host NoAH honeypots
- ~ problem: wide dark address space we could monitor vs limited number of honeypots

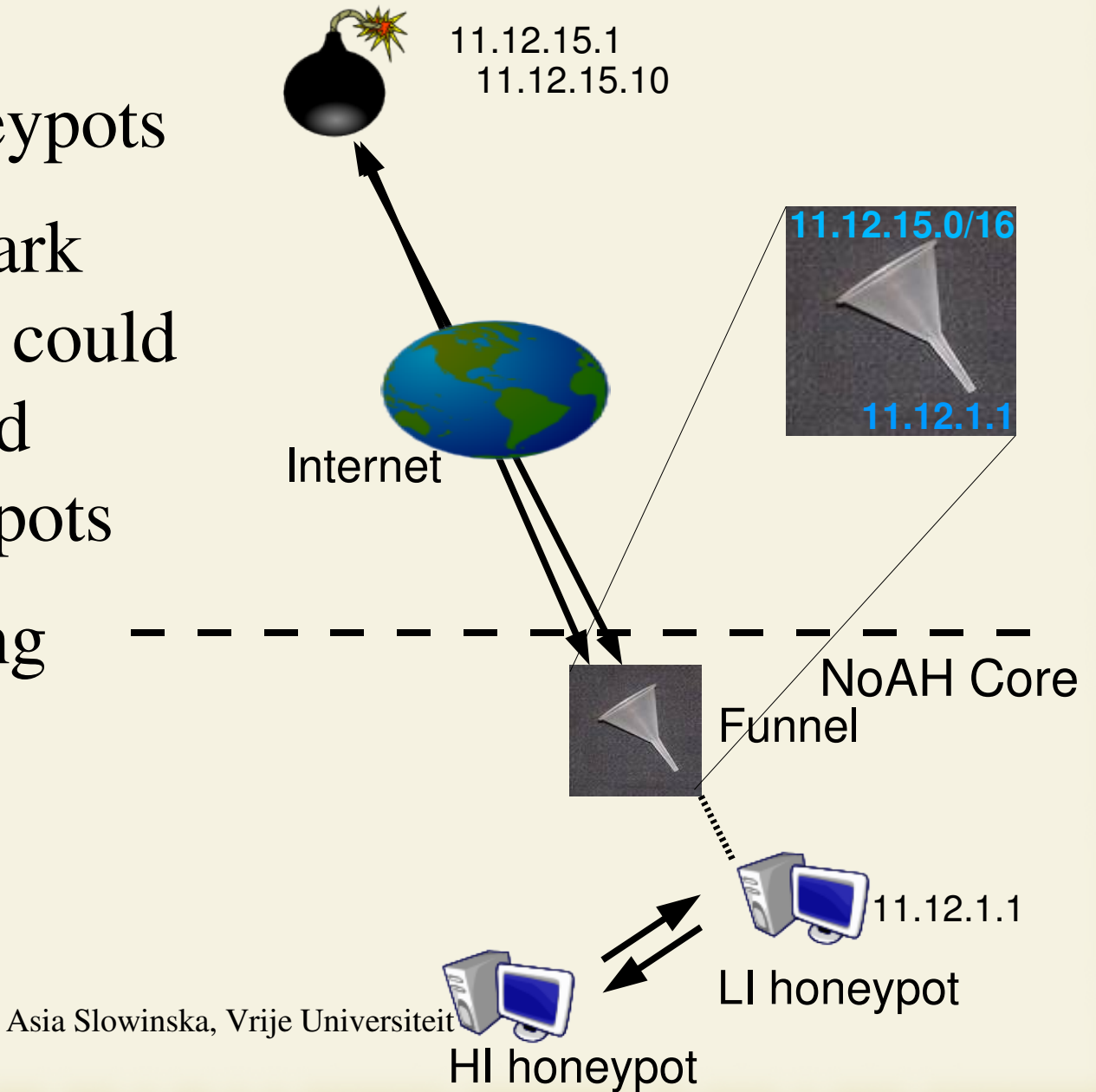
Core organizations

- ~ host NoAH honeypots
- ~ problem: wide dark address space we could monitor vs limited number of honeypots
- ~ solution: funelling



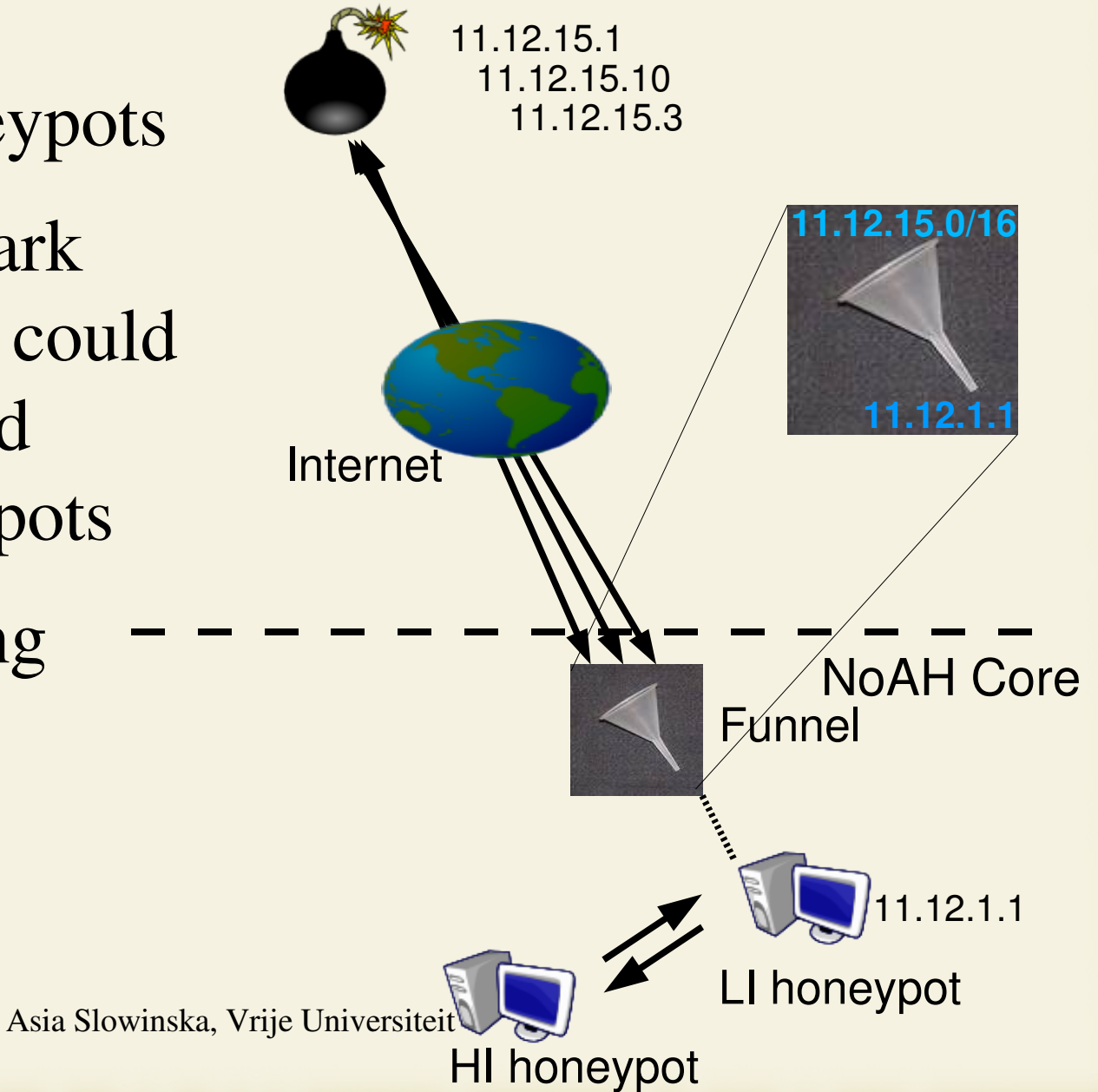
Core organizations

- ~ host NoAH honeypots
- ~ problem: wide dark address space we could monitor vs limited number of honeypots
- ~ solution: funelling



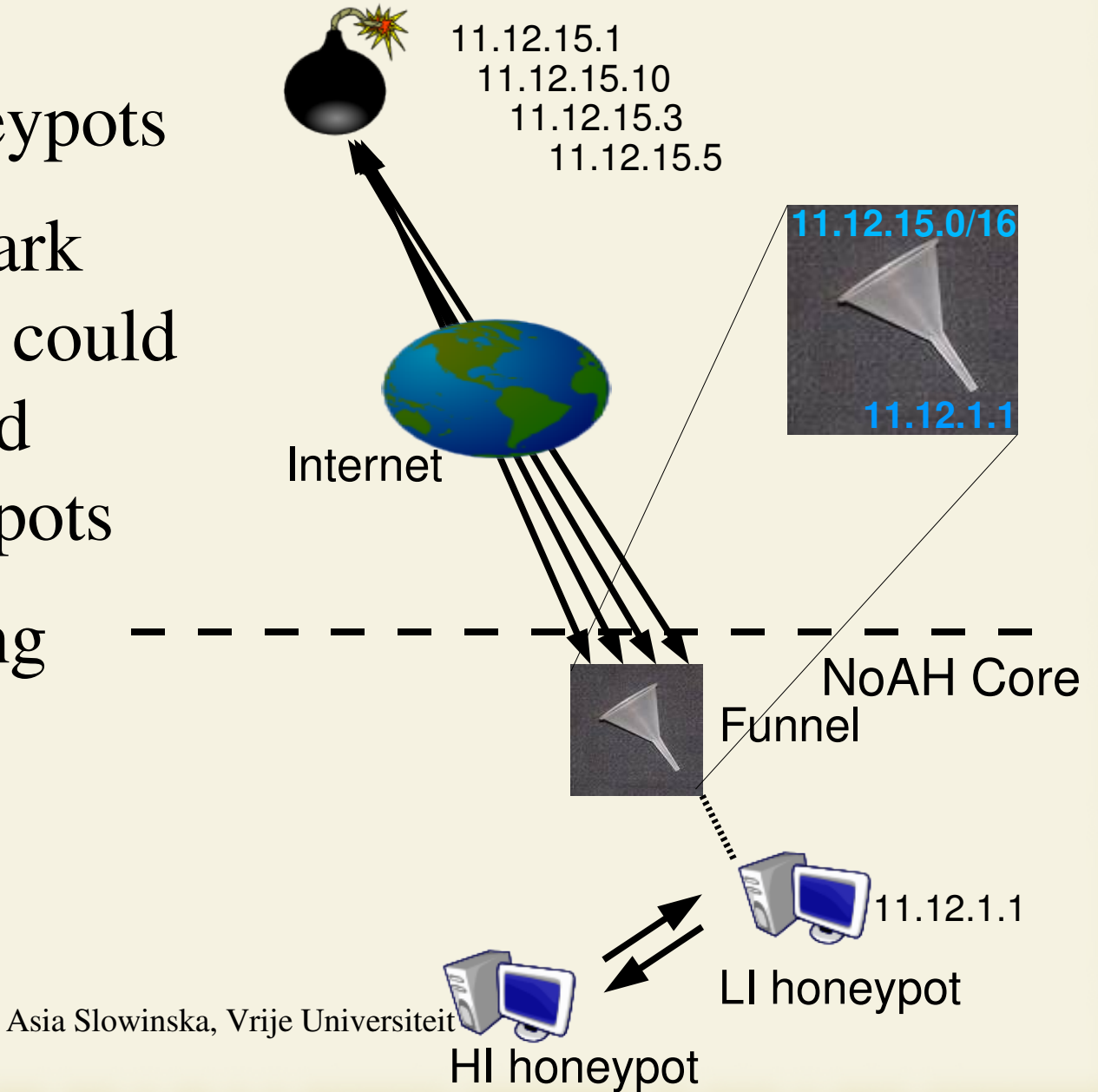
Core organizations

- ~ host NoAH honeypots
- ~ problem: wide dark address space we could monitor vs limited number of honeypots
- ~ solution: funelling

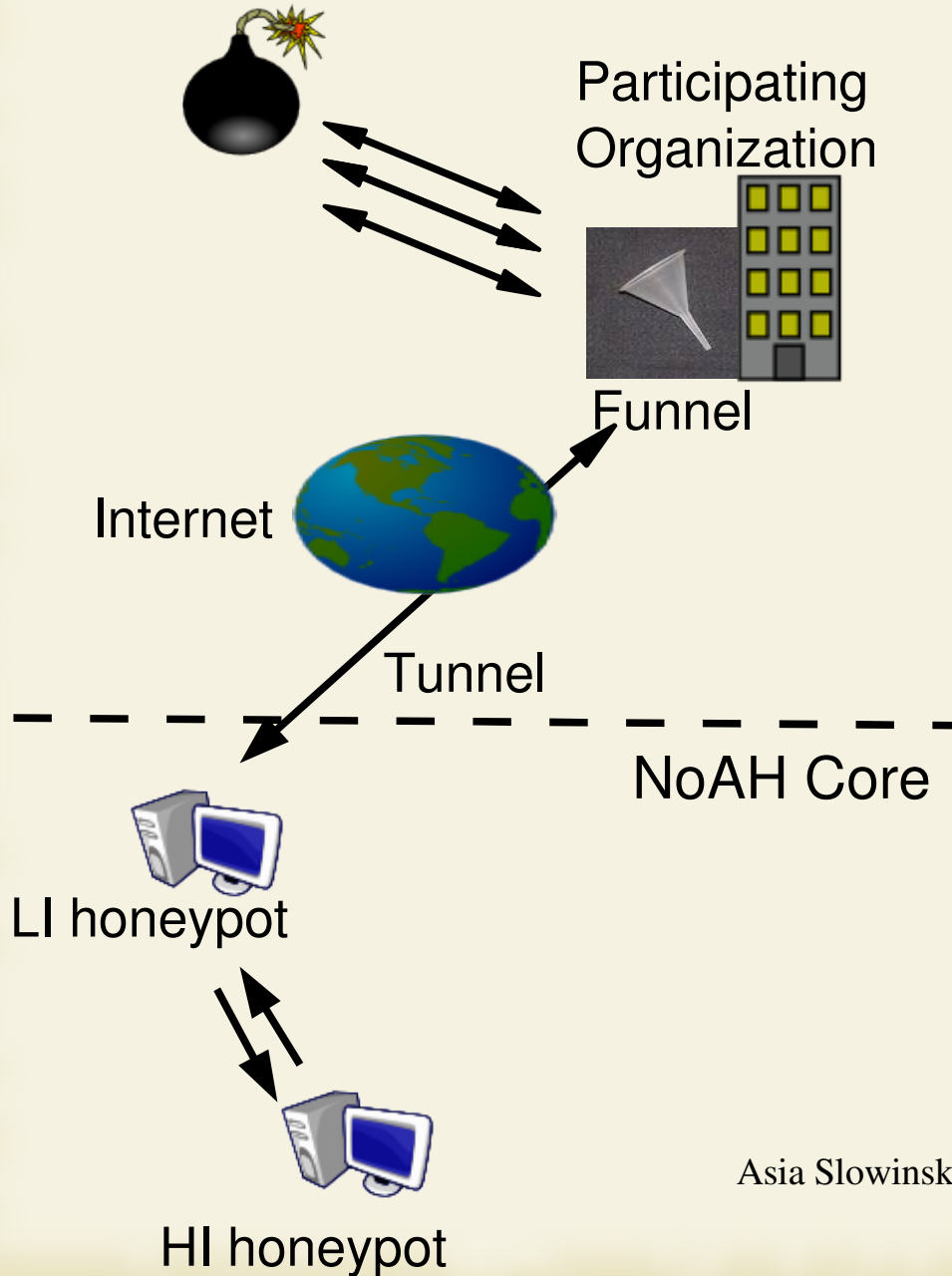


Core organizations

- ~ host NoAH honeypots
- ~ problem: wide dark address space we could monitor vs limited number of honeypots
- ~ solution: funelling



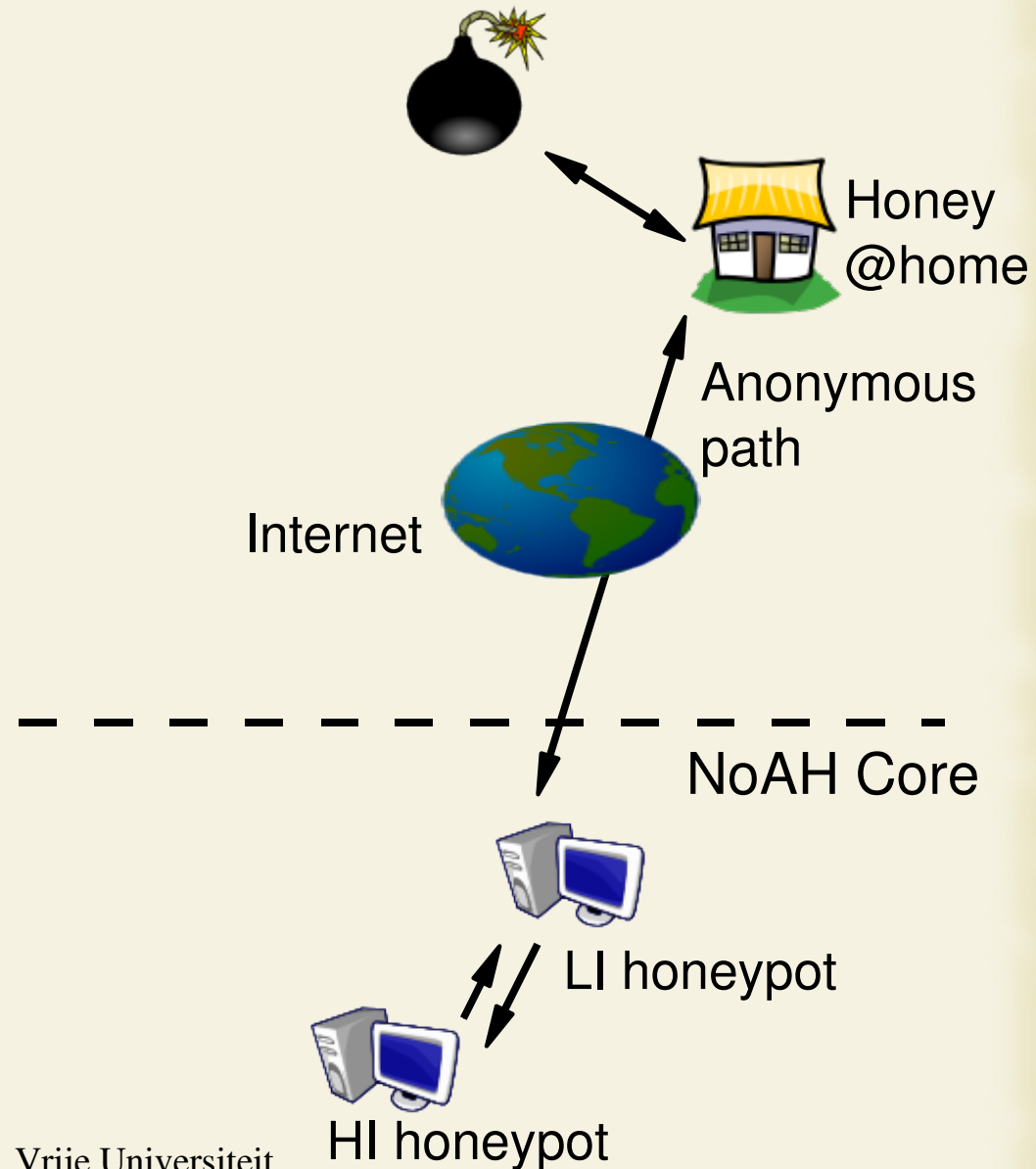
Cooperating organizations



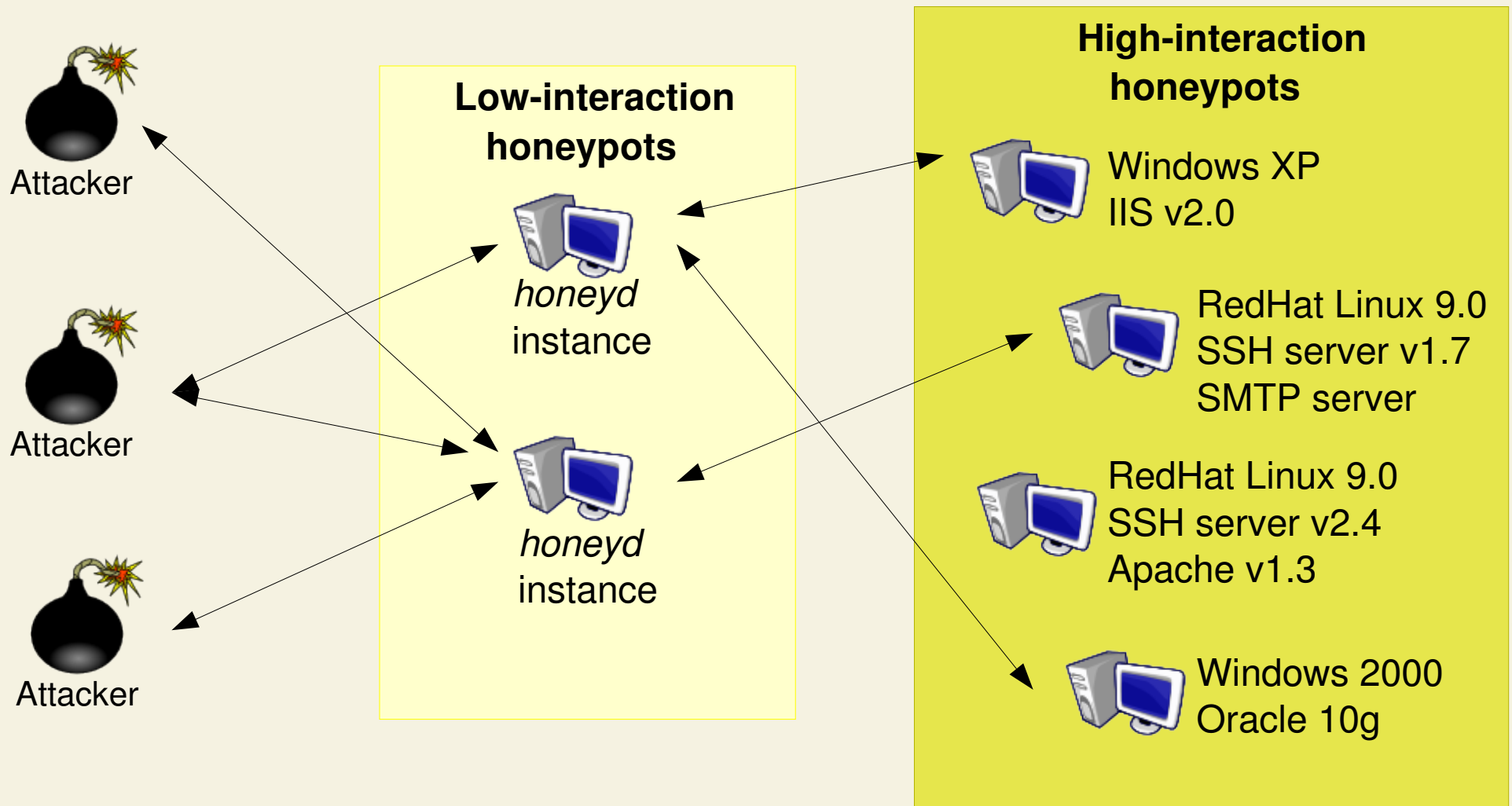
- ~ do not maintain NoAH honeypots
- ~ traffic arriving at the dark space is redirected to the NoAH core
- ~ install and run funnel component only

Honey@home

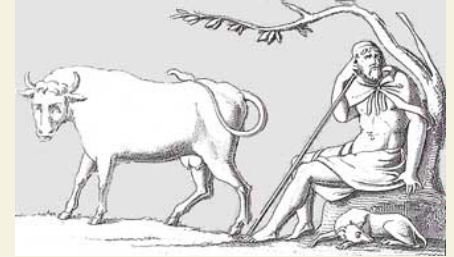
- ~ homes or small offices
- ~ a honeypot daemon running in the background
- ~ easy to install
- ~ dark space
 - ~ unused IP addresses
 - ~ unused TCP/UDP ports (or a subset of them)
- ~ forwards all traffic for the dark space to the NoAH core via an anonymous path



Cooperation between LI and HI honeypots

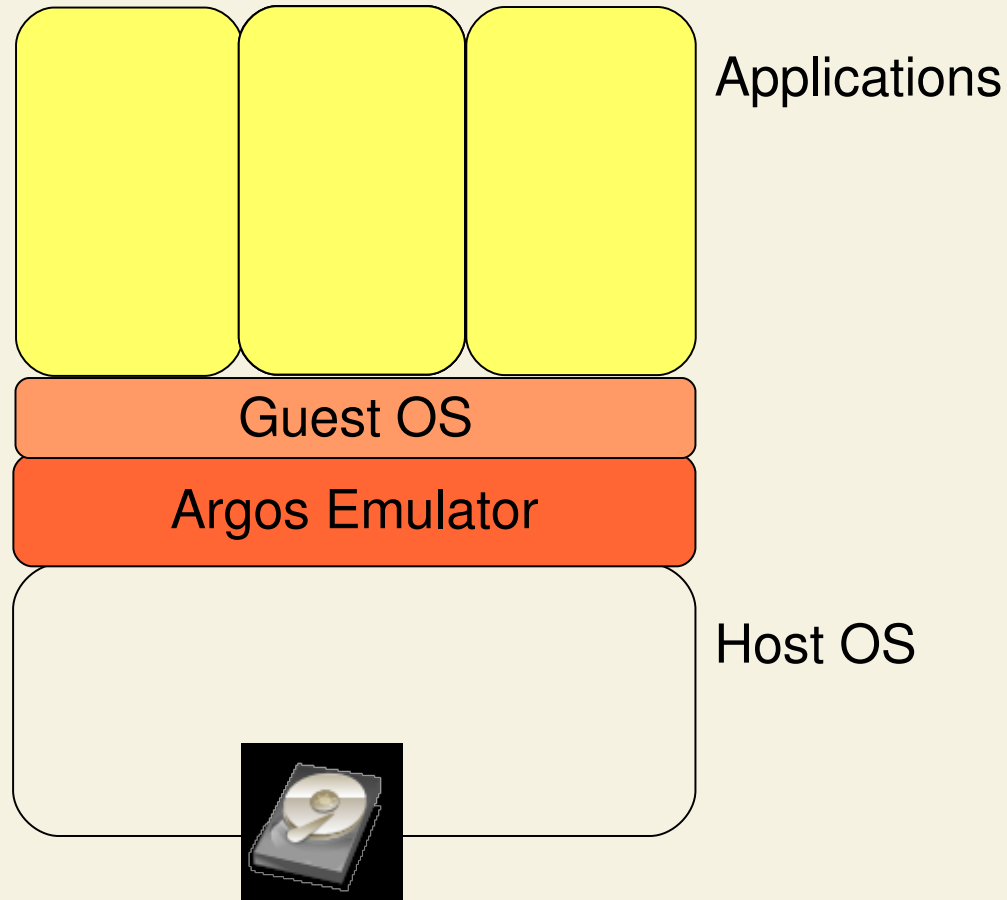
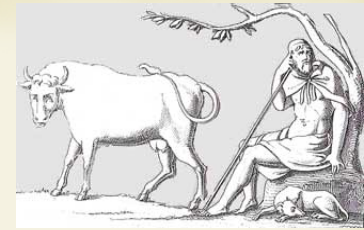


High-interaction honeypots in NoAH – Argos

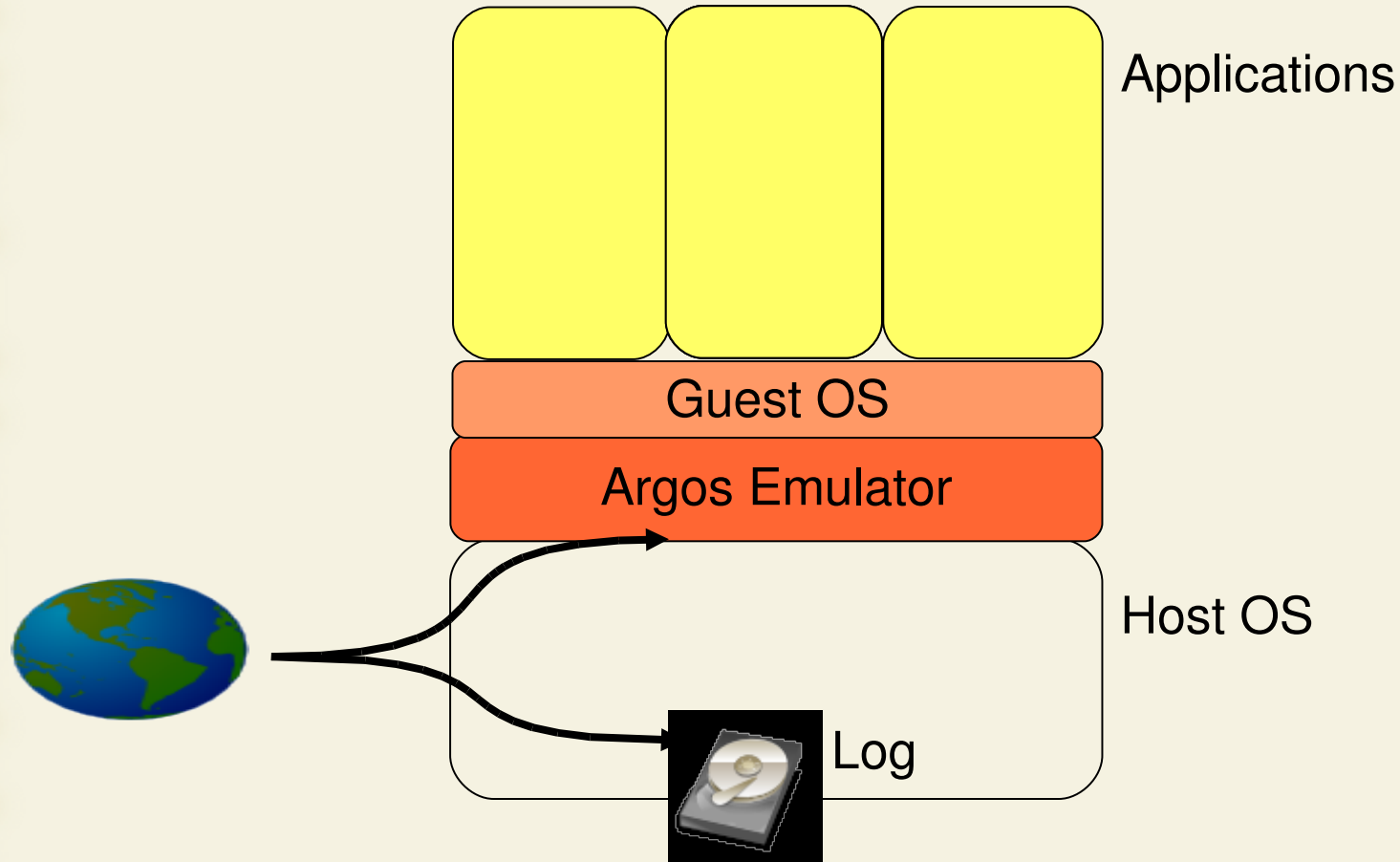
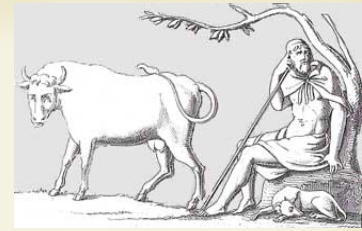


- ~ based on Qemu, an emulator
- ~ protects multiple OSs and all applications without modification
- ~ employs dynamic taint analysis
- ~ detects attacks that divert conventional control flow, e.g., exploits for buffer overflows, format strings, and double-free vulnerabilities

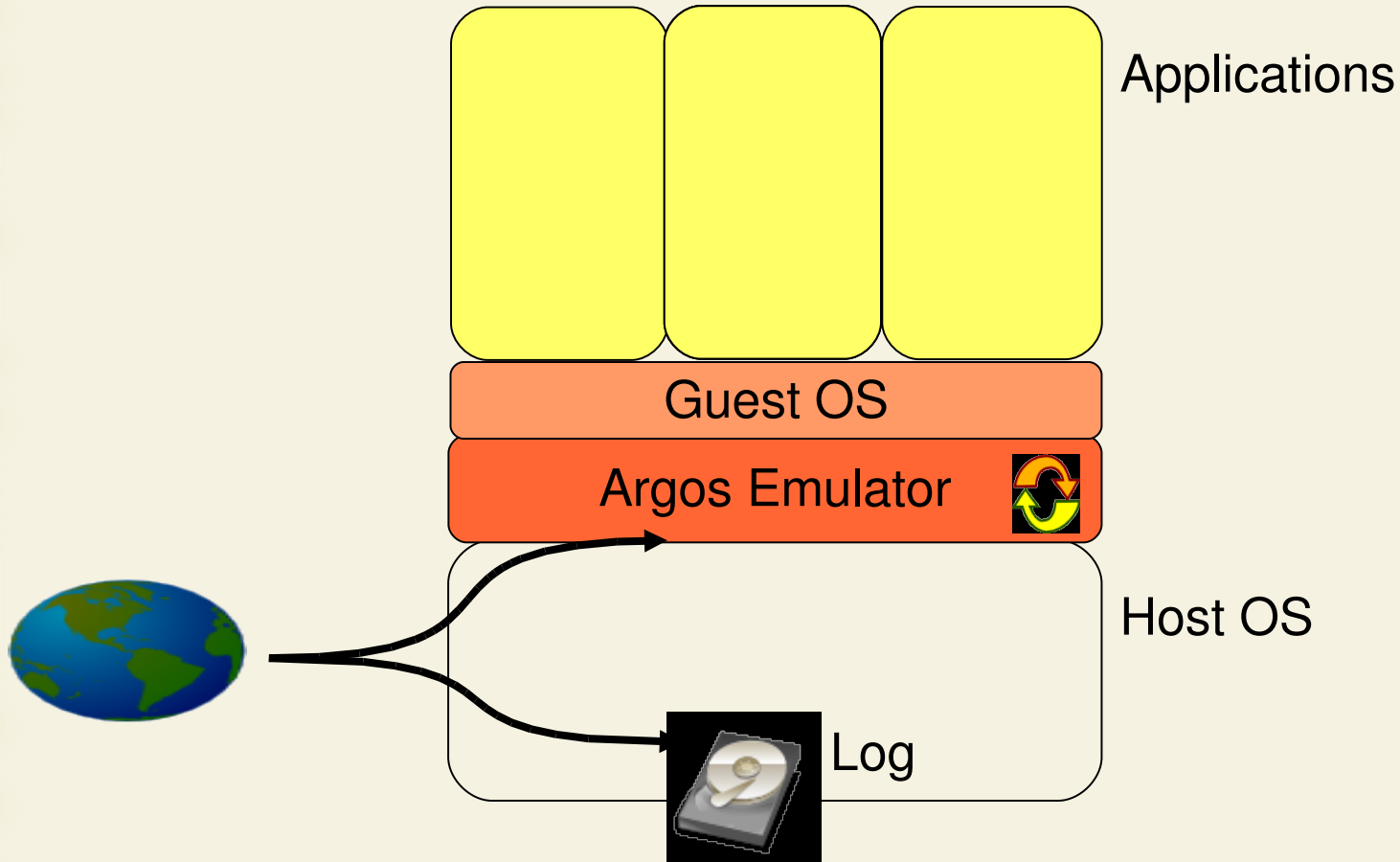
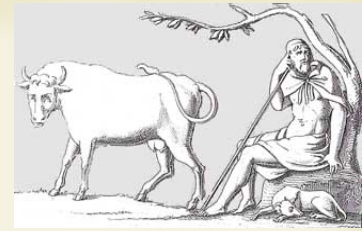
Argos design

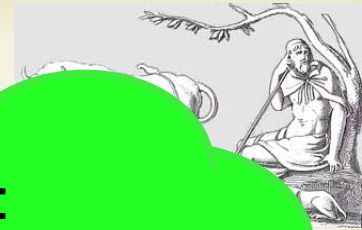


Argos design



Argos design

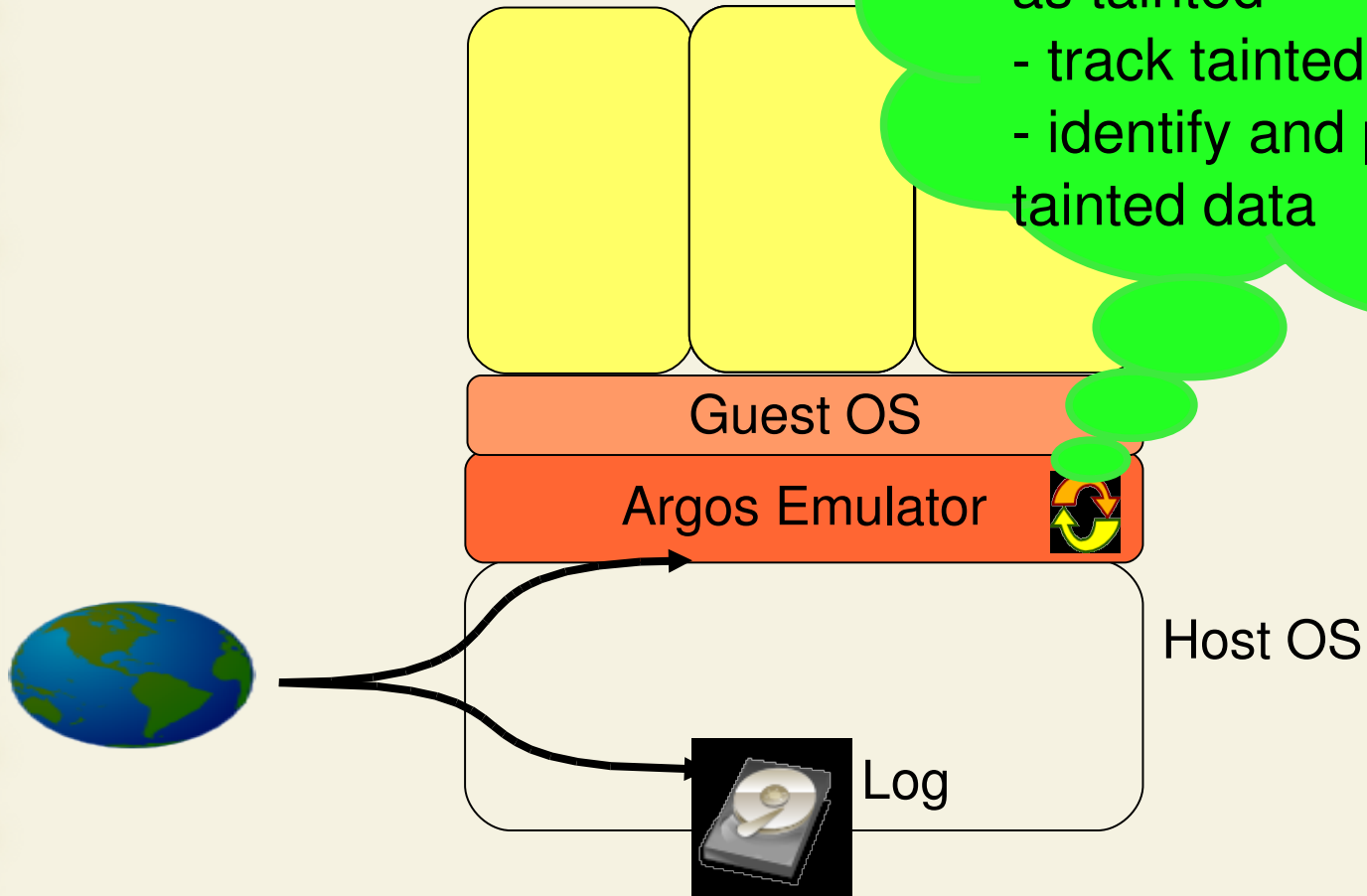




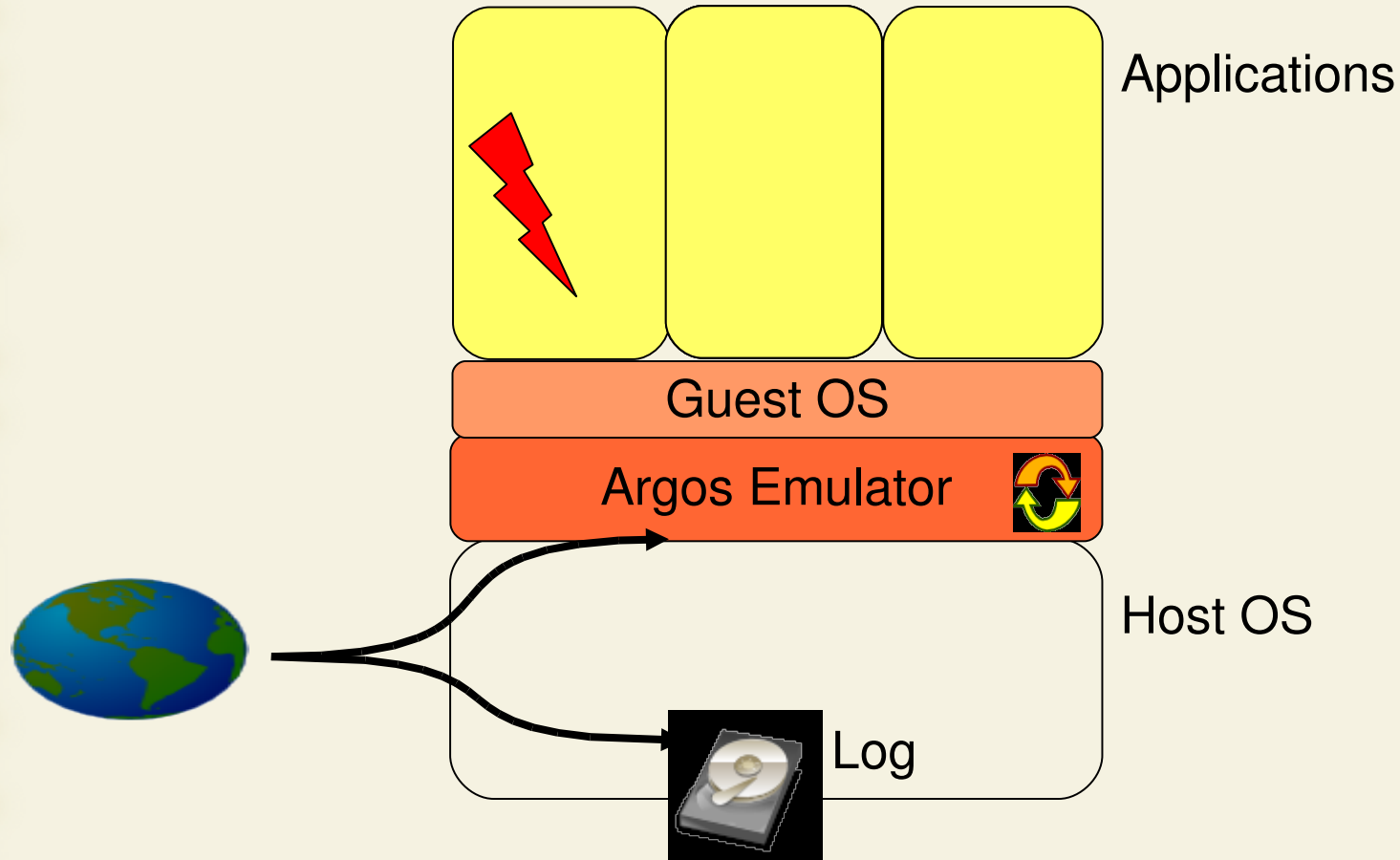
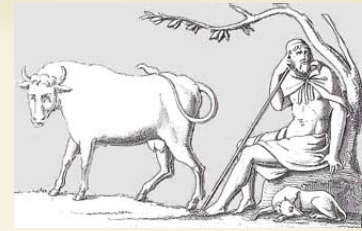
Argos d

Dynamic taint analysis:

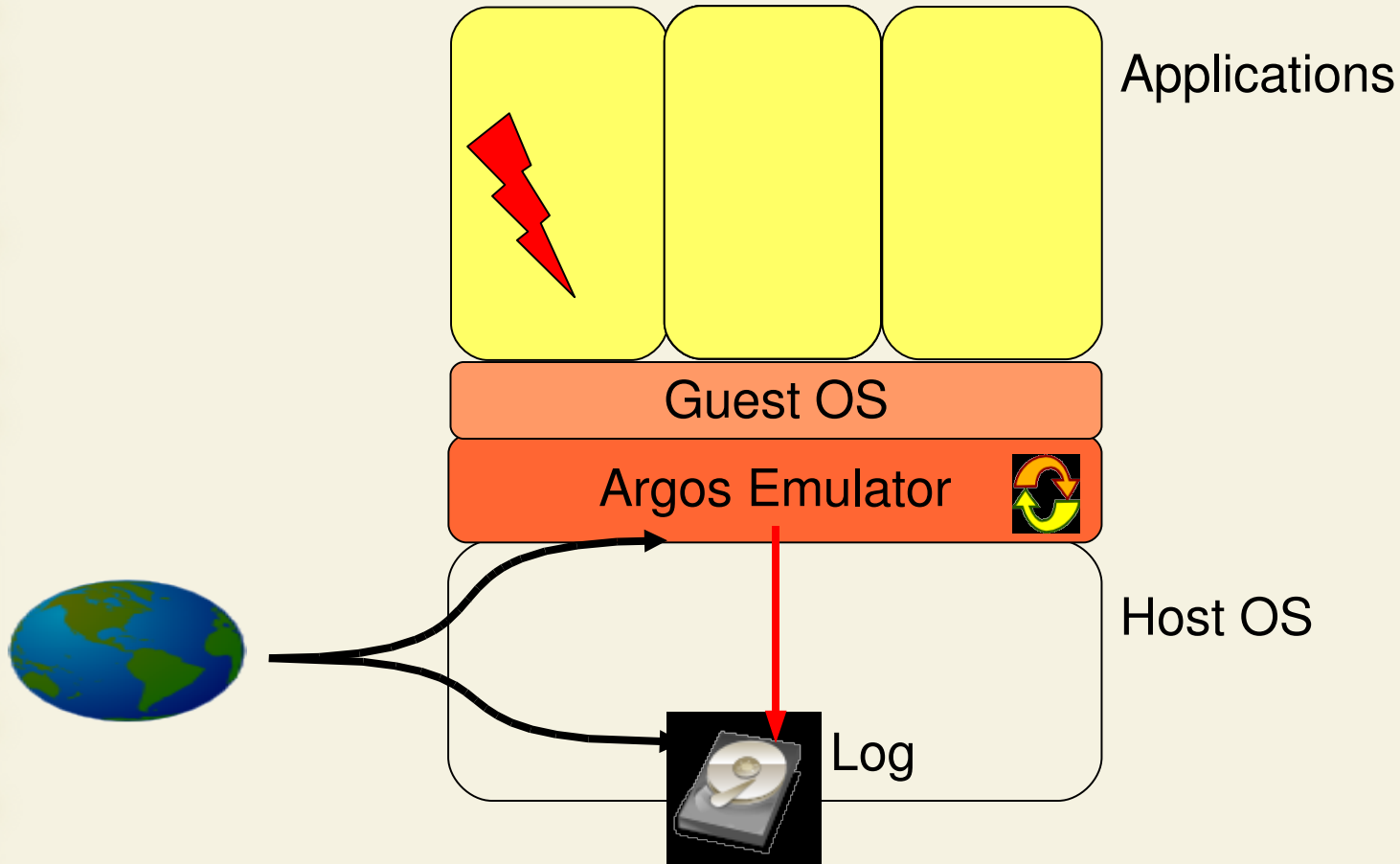
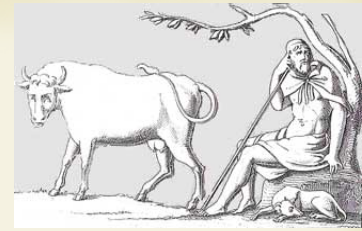
- tag data coming from the network as tainted
- track tainted data during execution
- identify and prevent usage of tainted data



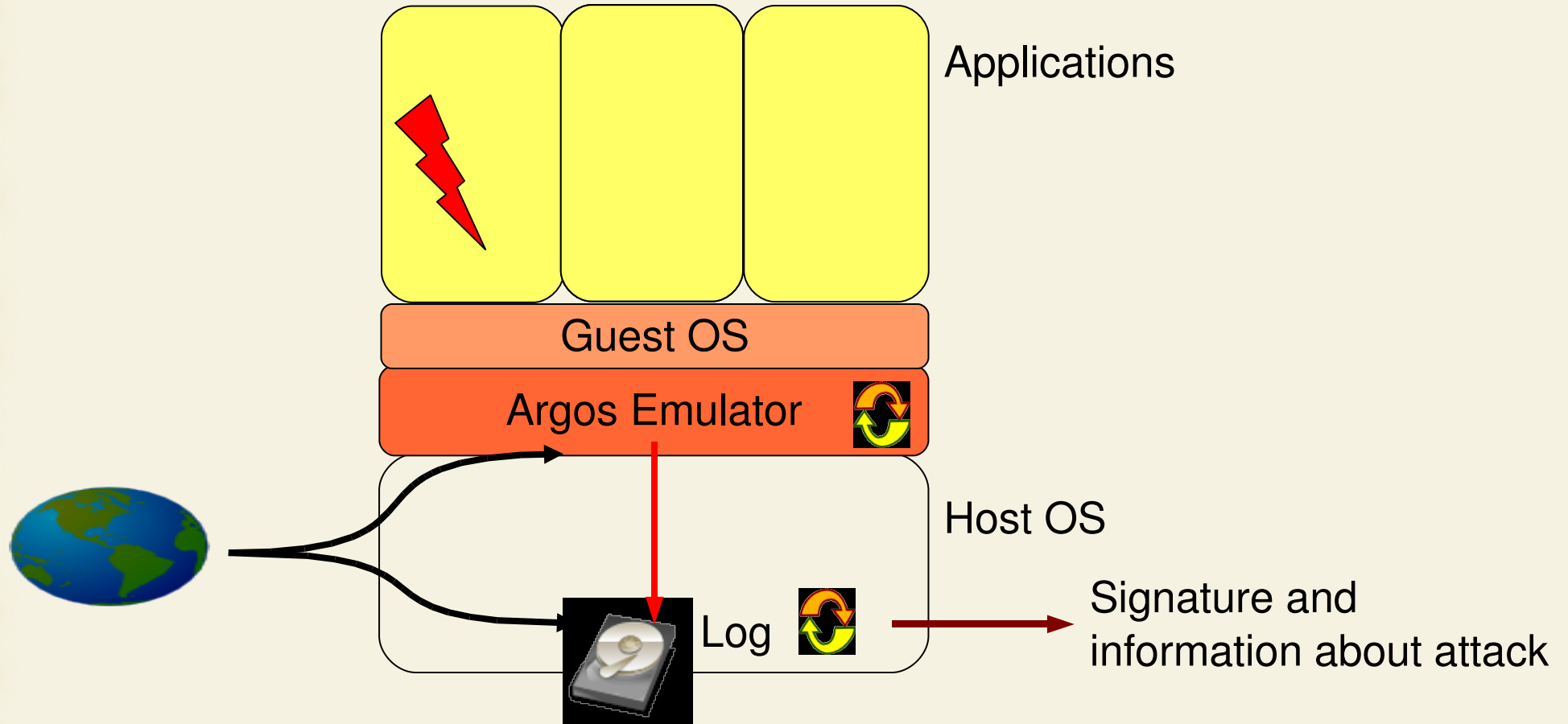
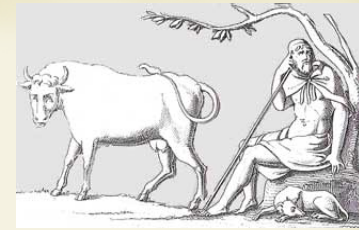
Argos design



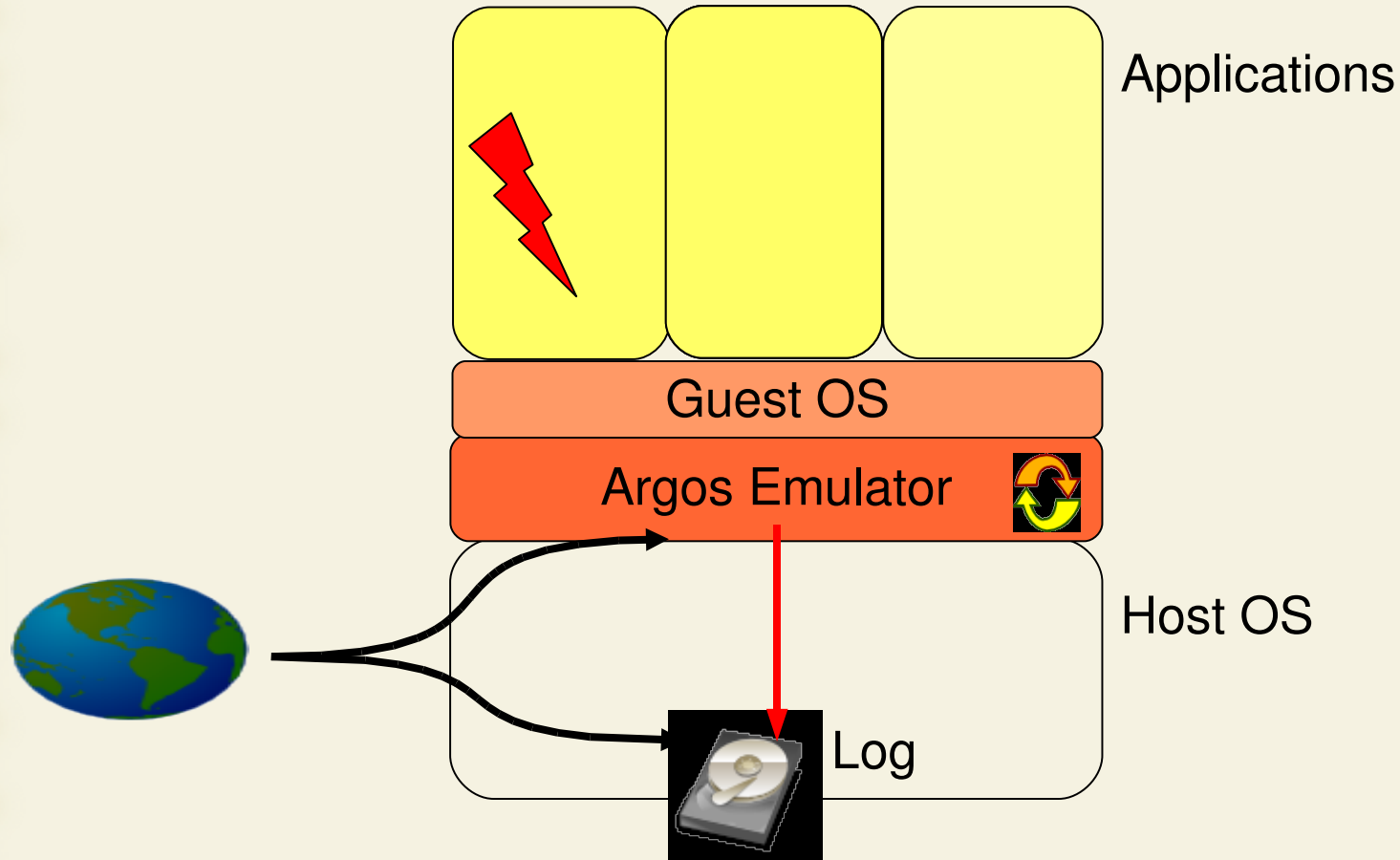
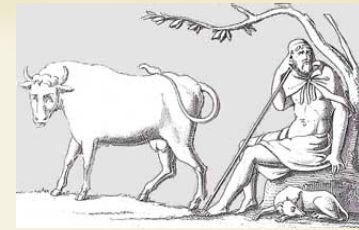
Argos design



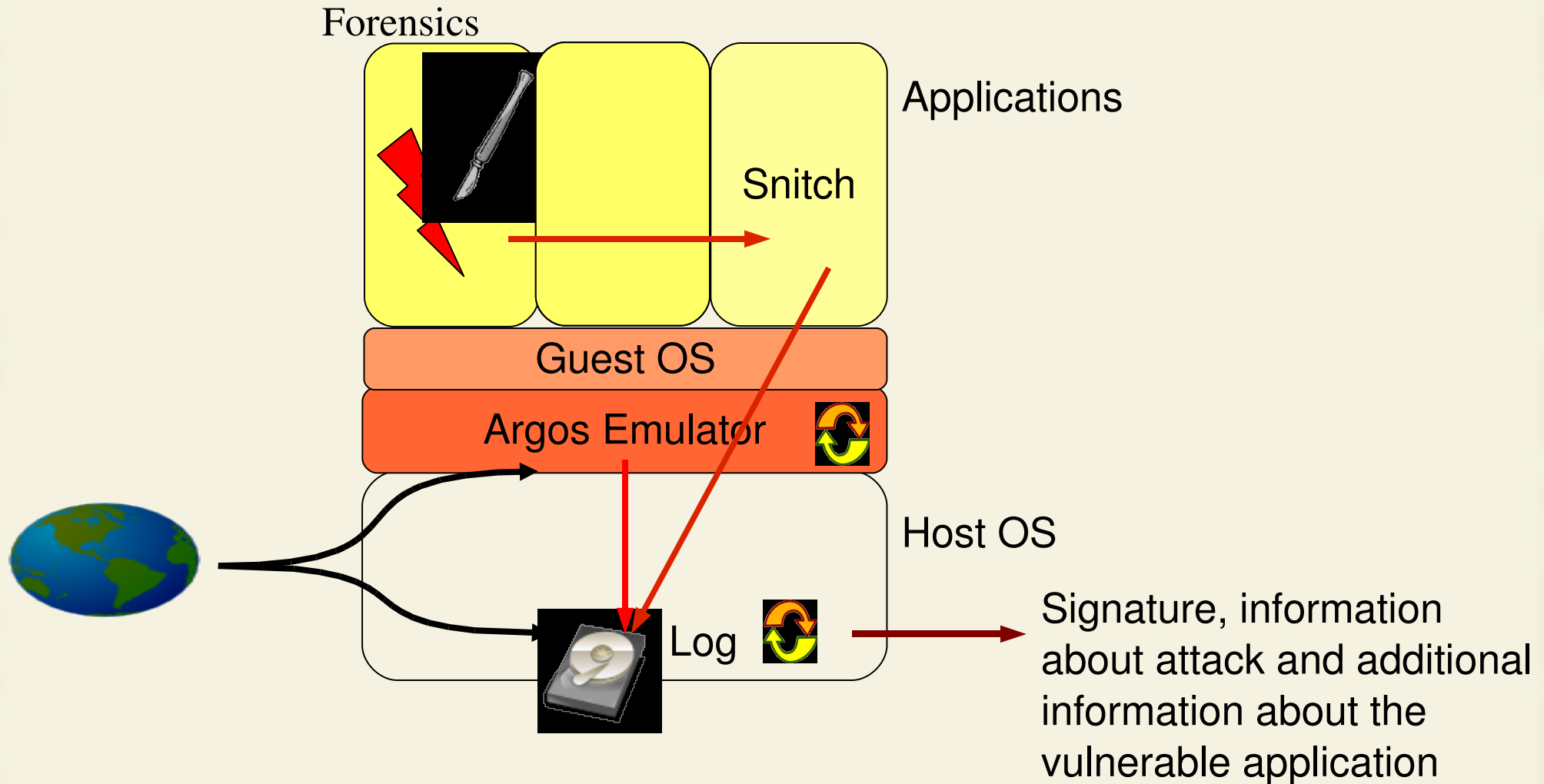
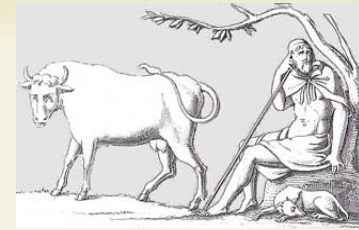
Argos design



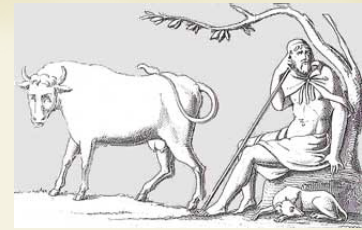
Forensics in Argos



Forensics in Argos

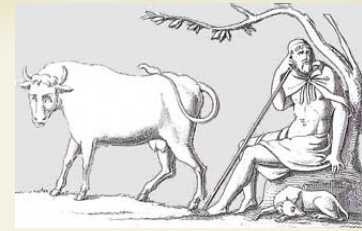


Attack detection



- ~ redirect control flow
 - ~ program counter must be loaded with a tainted value
 - ~ keep track of `call`, `jmp` and `ret` instructions
 - ~ check that the value loaded in program counter is not tainted
- ~ code-injection attacks
 - ~ format string attacks do not overwrite program counter with a tainted value
 - ~ check that the memory pointed by the value loaded in program counter is not tainted

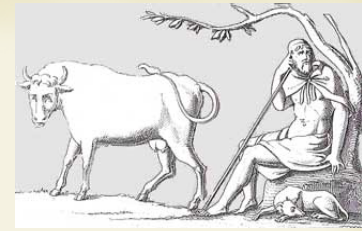
Argos: simple example



```
int main(int argc, char **argv)
{
    if (argc > 1) read_url(argv[1]);
    return 1;
}

int read_url(char *request)
{
    char url[100];
    if (!strncmp(request, "GET ", 4))
        strcpy(url, request + 4);
    return 1;
}
```

Argos: simple example

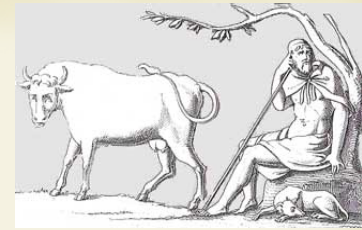


```
int main(int argc, char **argv)
{
    if (argc > 1) read_url(argv[1]);
    return 1;
}
```

```
int read_url(char *request)
{
    char url[100];
    if (!strncmp(request, "GET ", 4))
        strcpy(url, request + 4);
    return 1;
}
```

./test GET A....AAAAA<nasty_address>\0
 ↔
 100 chars

Argos: simple example

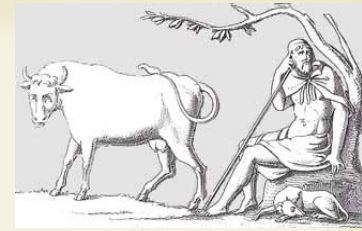


```
int main(int argc, char **argv)
{
    → if (argc > 1) read_url(argv[1]);
    return 1;
}
```

```
int read_url(char *request)
{
    char url[100];
    if (!strncmp(request, "GET ", 4))
        strcpy(url, request + 4);
    return 1;
}
```

```
./test GET A....AAAAA<nasty_address>\0
      ↕↔
    100 chars
```

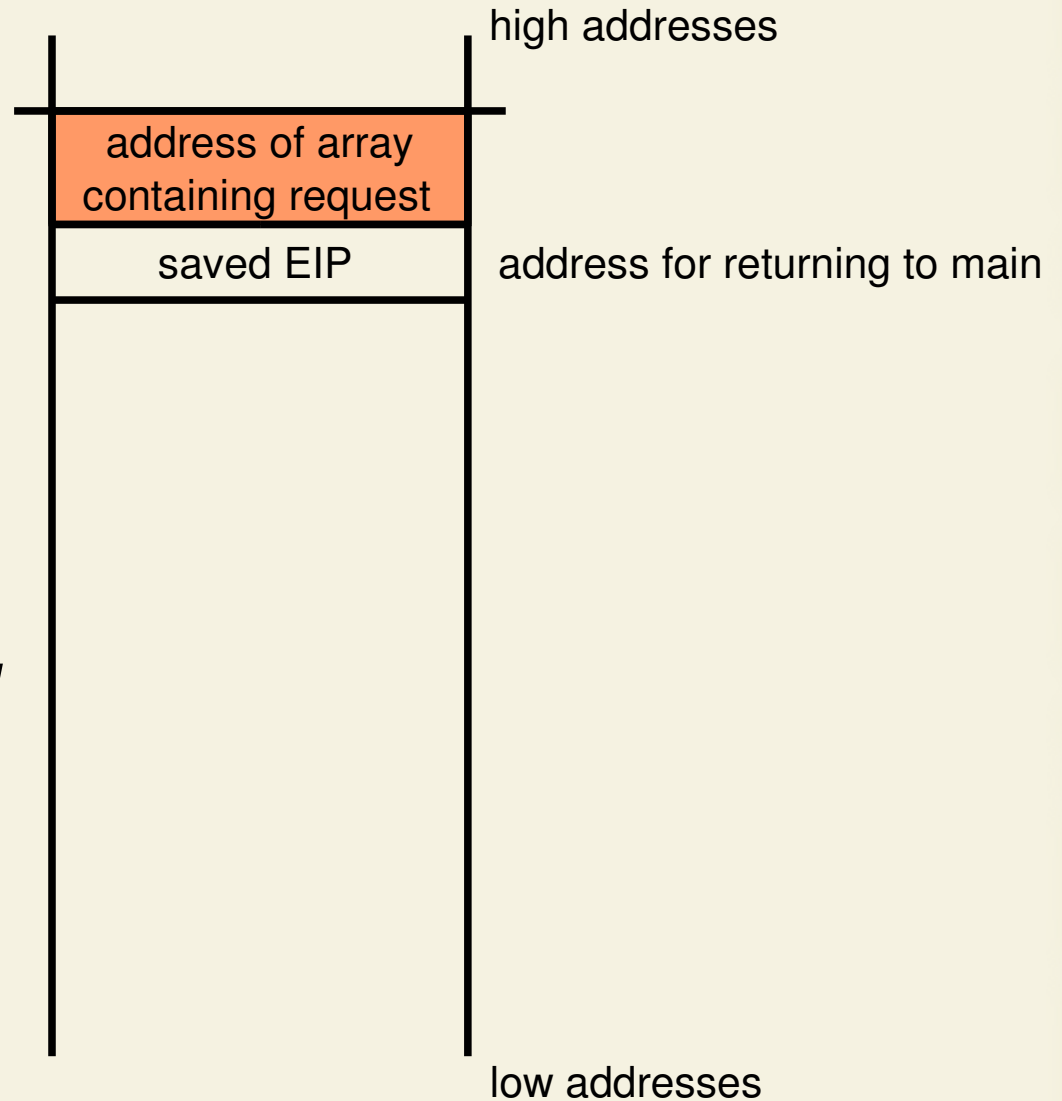
Argos: simple example



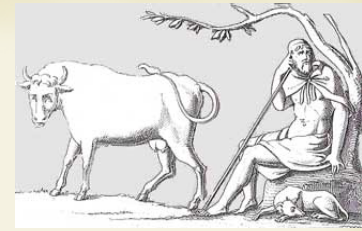
```
int main(int argc, char **argv)
{
    if (argc > 1) read_url(argv[1]);
    return 1;
}
```

```
int read_url(char *request)
{
    char url[100];
    if (!strncmp(request, "GET ", 4))
        strcpy(url, request + 4);
    return 1;
}
```

./test GET A...AAAAA<nasty_address>\0
 ↔
 100 chars



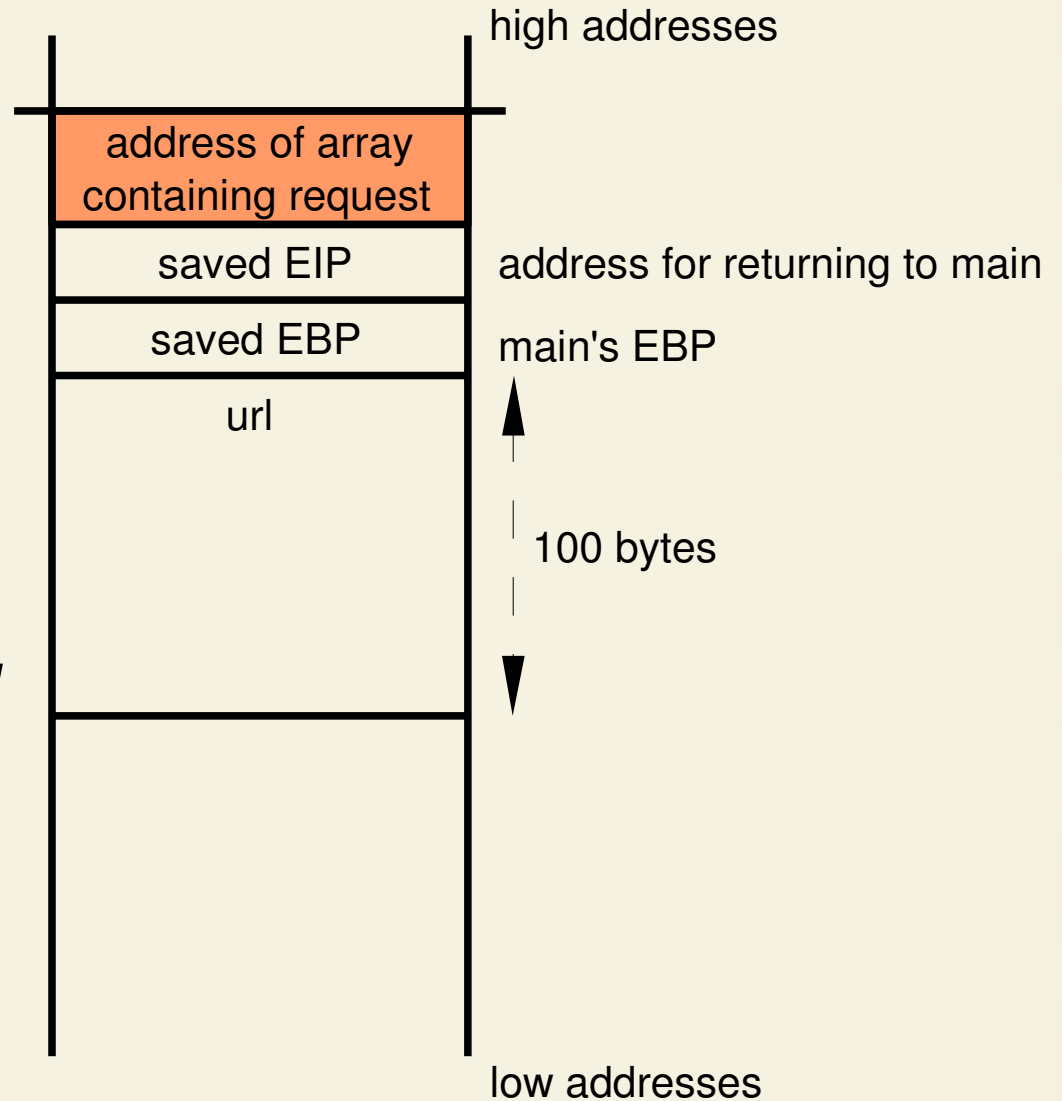
Argos: simple example



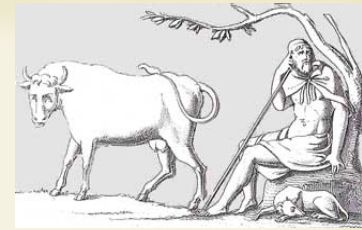
```
int main(int argc, char **argv)
{
    if (argc > 1) read_url(argv[1]);
    return 1;
}
```

```
int read_url(char *request)
{
    char url[100];
    if (!strncmp(request, "GET ", 4))
        strcpy(url, request + 4);
    return 1;
}
```

./test GET A....AAAAA<nasty_address>\0
100 chars



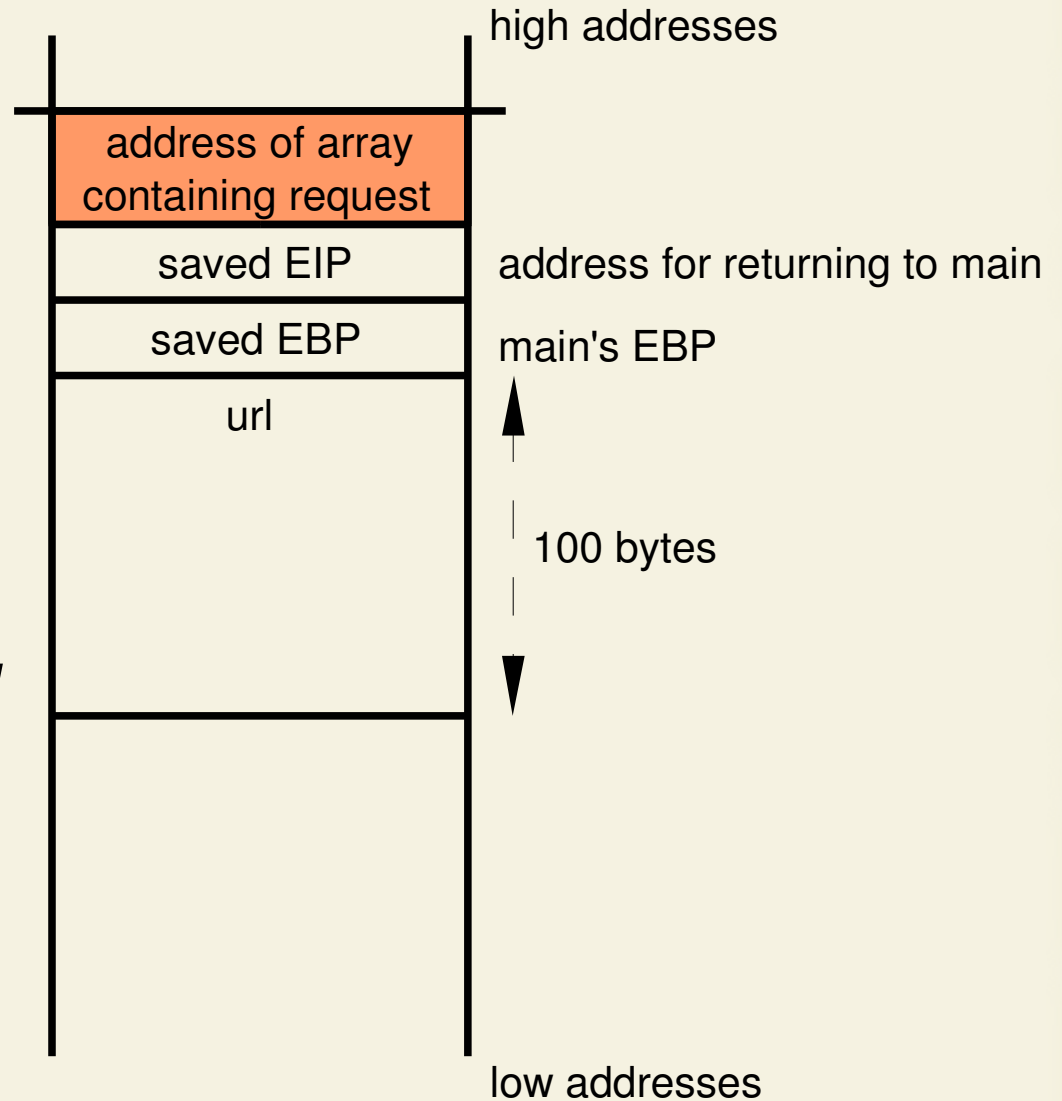
Argos: simple example



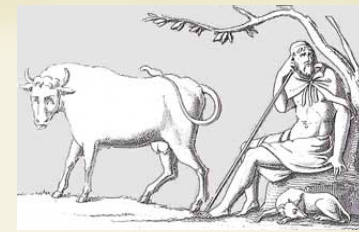
```
int main(int argc, char **argv)
{
    if (argc > 1) read_url(argv[1]);
    return 1;
}
```

```
int read_url(char *request)
{
    char url[100];
    if (!strncmp(request, "GET ", 4))
        strcpy(url, request + 4);
    return 1;
}
```

./test GET A...AAAAA<nasty_address>\0
 ↔
 100 chars



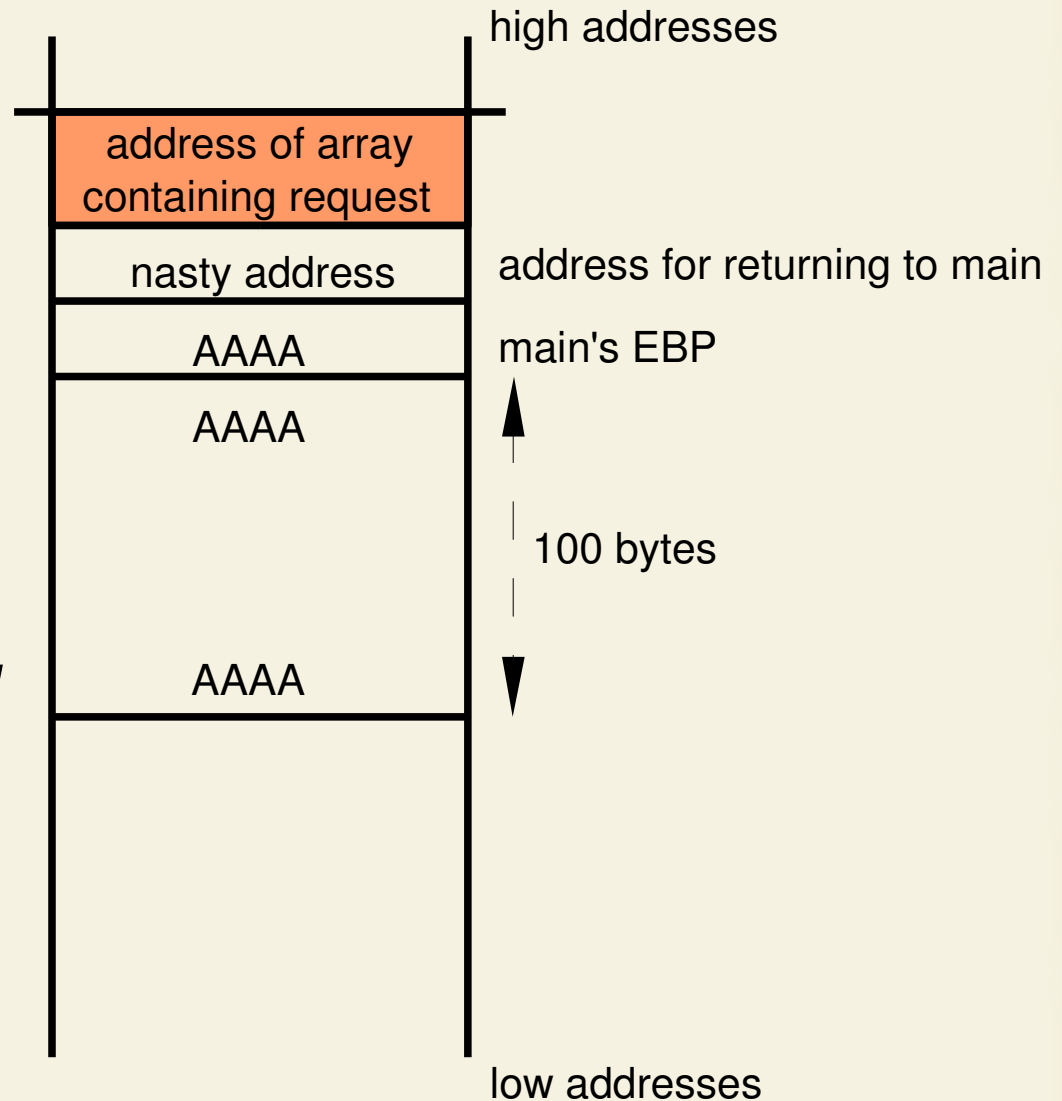
Argos: simple example



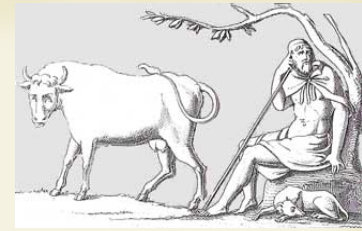
```
int main(int argc, char **argv)
{
    if (argc > 1) read_url(argv[1]);
    return 1;
}
```

```
int read_url(char *request)
{
    char url[100];
    if (!strncmp(request, "GET ", 4))
        strcpy(url, request + 4);
    return 1;
}
```

./test GET A...AAAAA<nasty_address>\0
 ↔
 100 chars



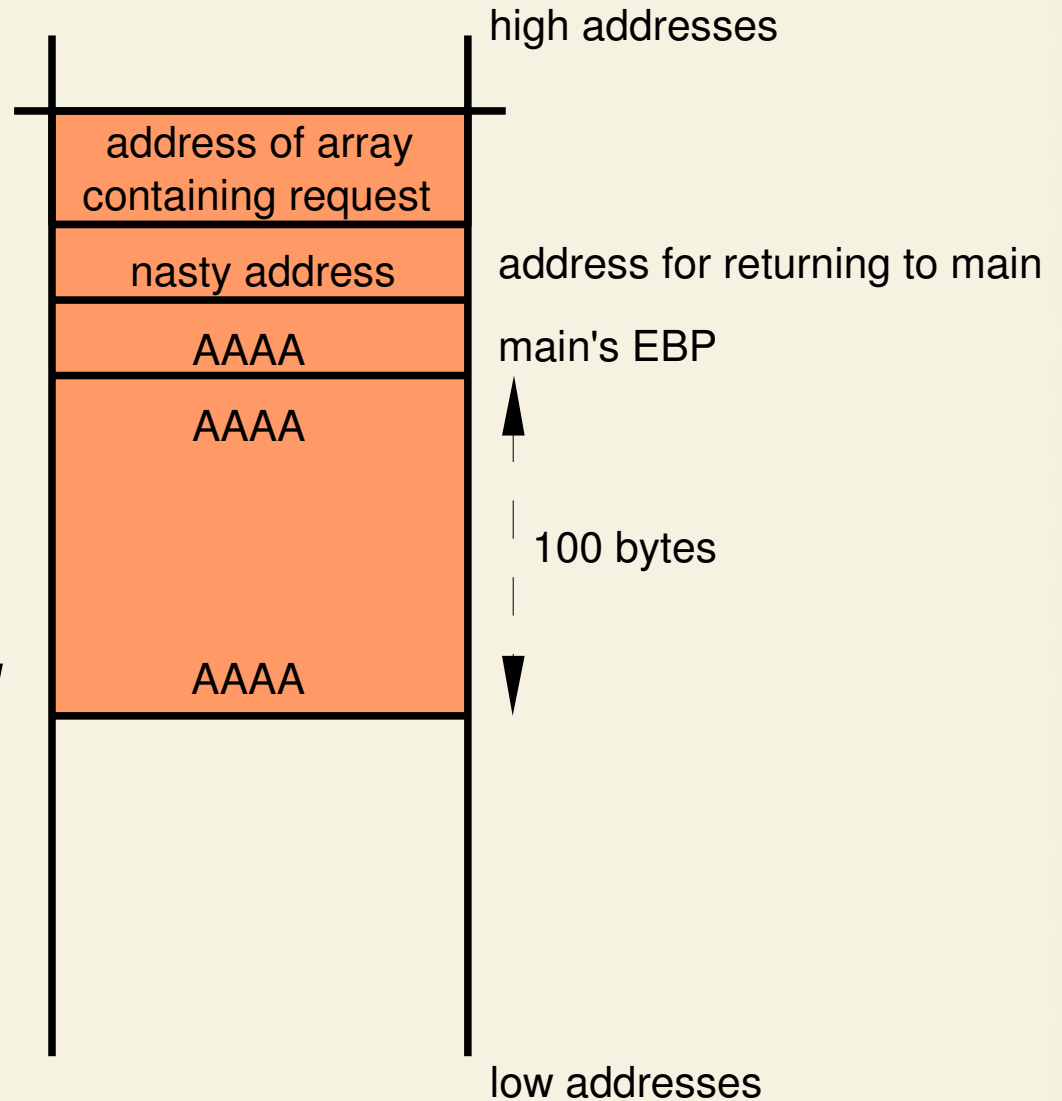
Argos: simple example



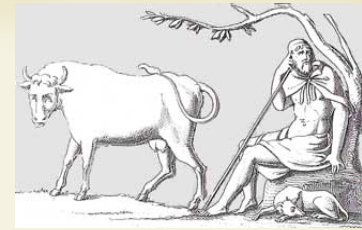
```
int main(int argc, char **argv)
{
    if (argc > 1) read_url(argv[1]);
    return 1;
}
```

```
int read_url(char *request)
{
    char url[100];
    if (!strncmp(request, "GET ", 4))
        strcpy(url, request + 4);
    return 1;
}
```

./test GET A...AAAAA<nasty_address>\0
100 chars



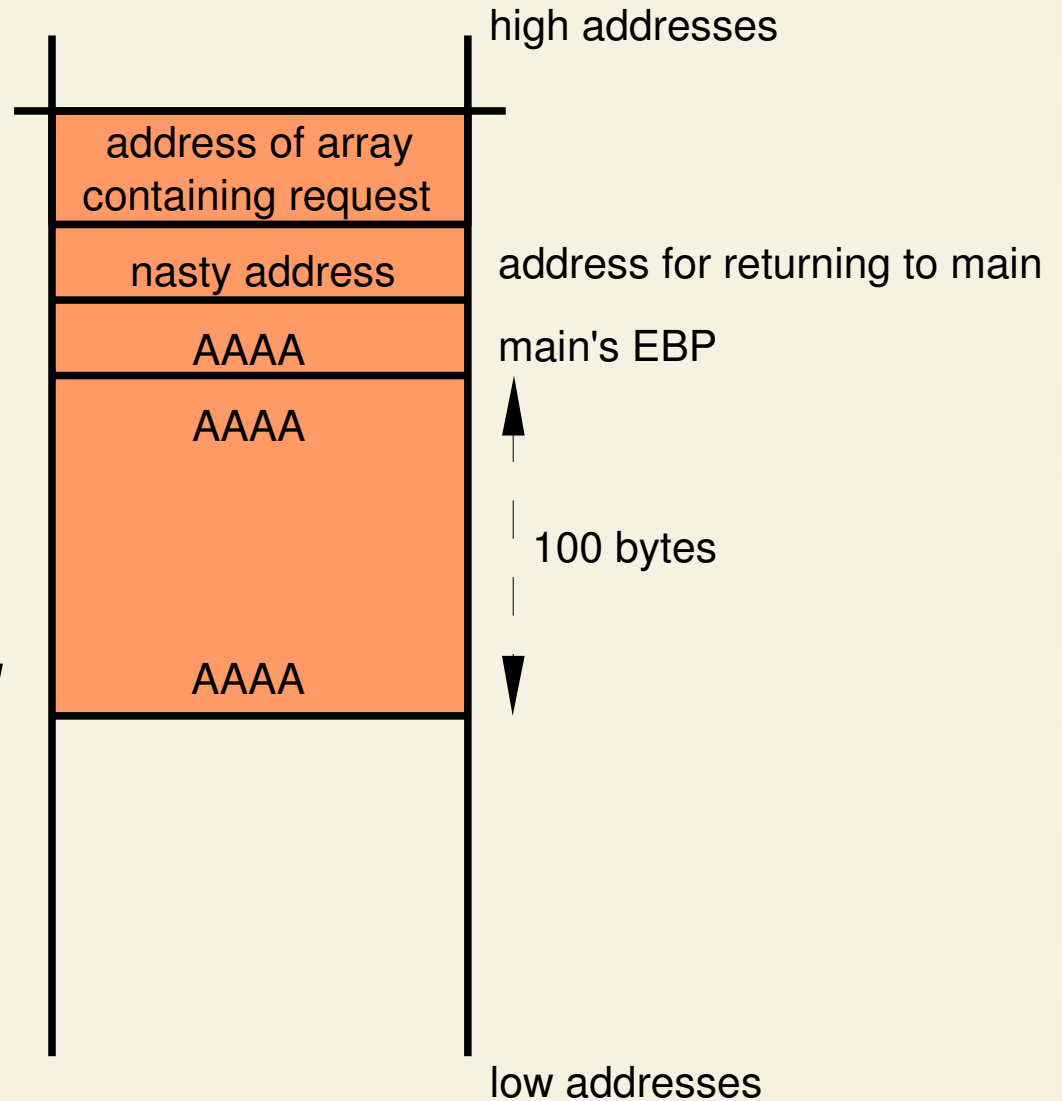
Argos: simple example



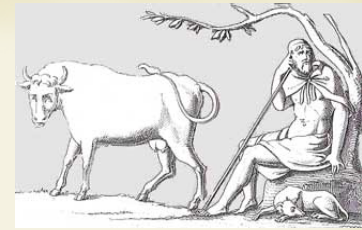
```
int main(int argc, char **argv)
{
    if (argc > 1) read_url(argv[1]);
    return 1;
}
```

```
int read_url(char *request)
{
    char url[100];
    if (!strncmp(request, "GET ", 4))
        strcpy(url, request + 4);
    return 1;
}
```

./test GET A...AAAAA<nasty_address>\0
100 chars



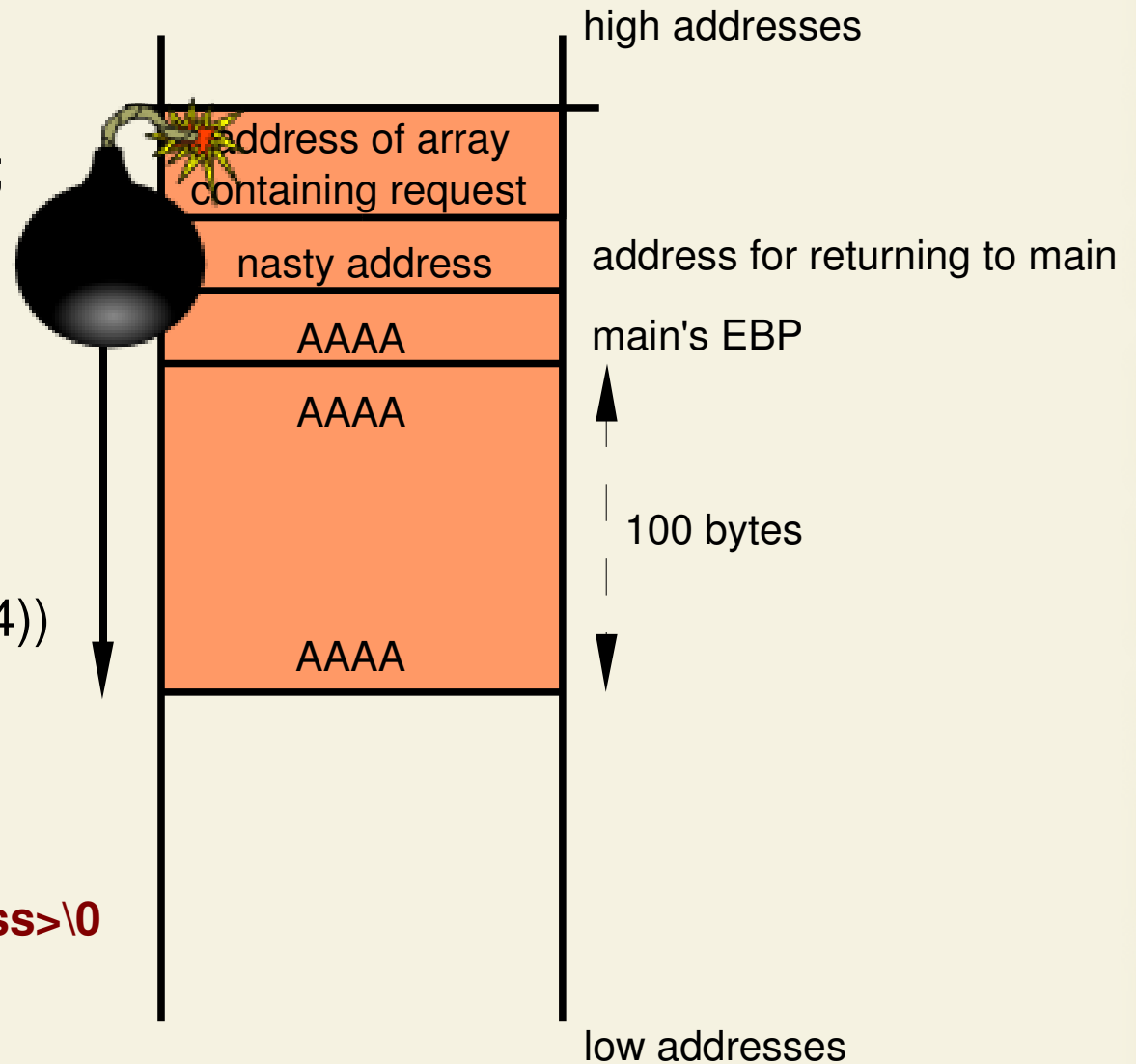
Argos: simple example



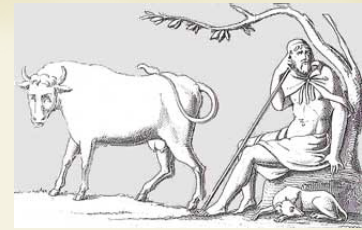
```
int main(int argc, char **argv)
{
    if (argc > 1) read_url(argv[1]);
    return 1;
}
```

```
int read_url(char *request)
{
    char url[100];
    if (!strncmp(request, "GET ", 4))
        strcpy(url, request + 4);
    return 1;
}
```

./test GET A...AAAAA<nasty_address>\0
100 chars



Security evaluation



Apache chunked encoding overflow

IIS ISAPI .printer host header overflow

WebDav ntdll.dll overflow

FrontPage Server Extensions Debug Overflow

War-FTP overflow

ASN.1 Library Bitstring Heap Overflow

Windows Message Queueing Remote Overflow

RPC DCOM Interface overflow

LSASS Overflow

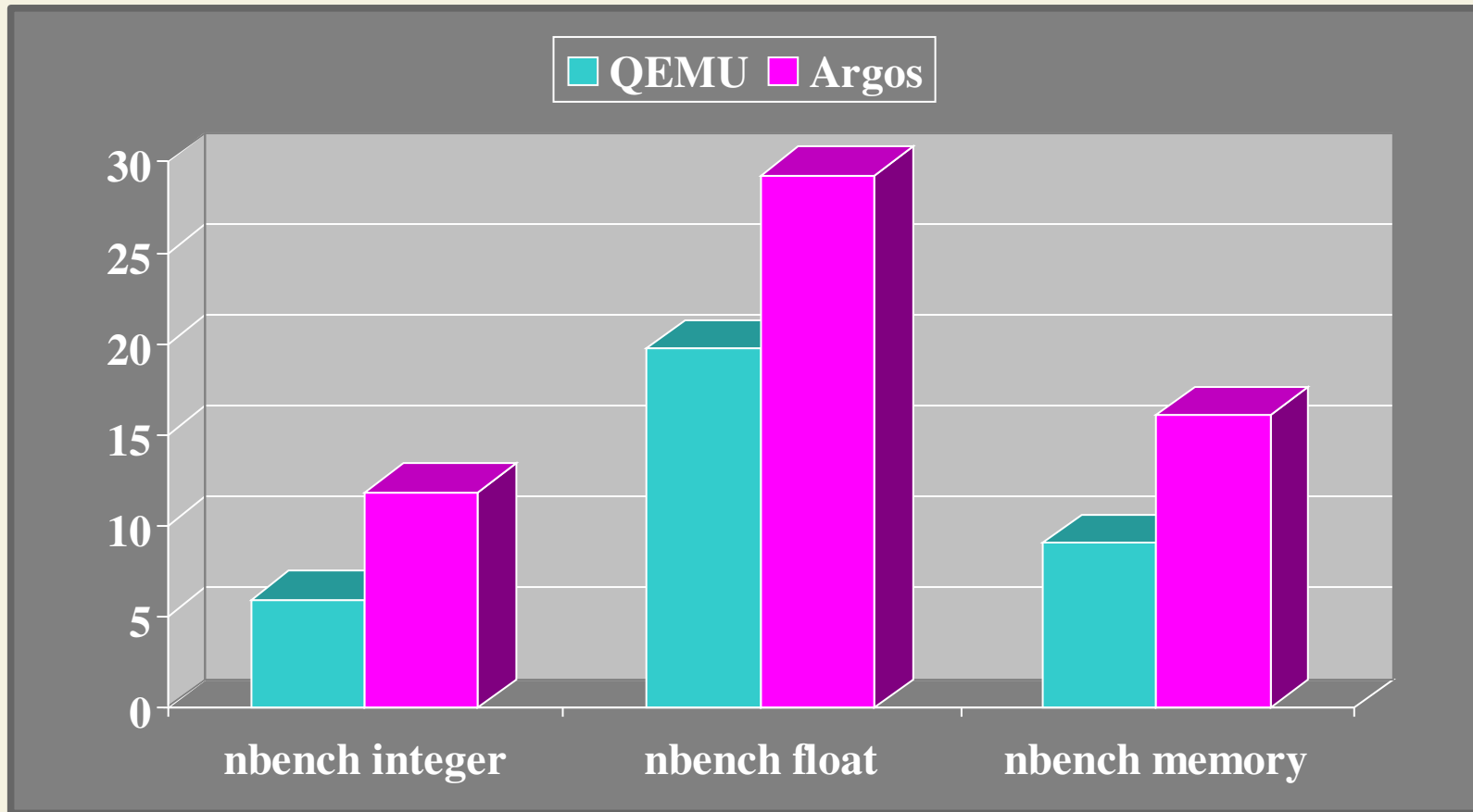
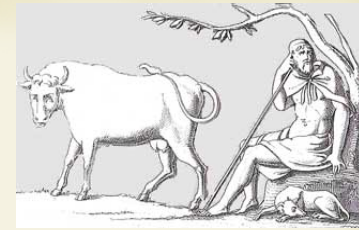
Windows PnP Service Remote Overflow

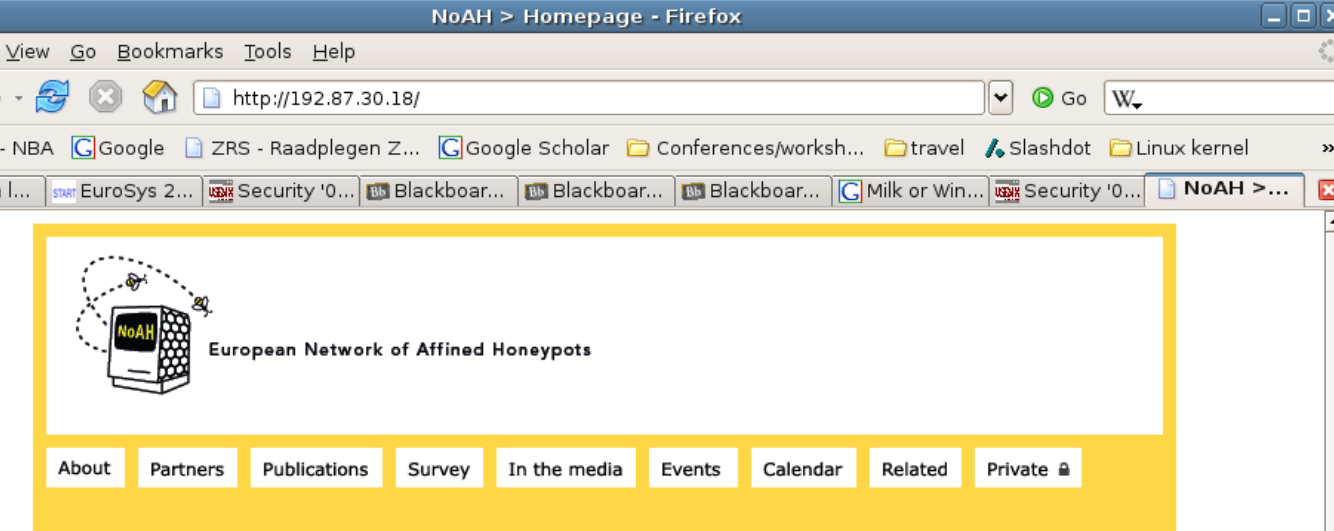
nbSMTP remote format string exploit

WMF exploit

Microsoft net API overflow MS06-040

Performance





http://www.few.vu.nl/argos

Welcome to NoAH

NoAH is a Specific Support Action in the Sixth Framework Programme of the European Union.

The project aims to gather and analyse information about the nature of Internet cyberattacks. It will also develop an infrastructure to detect and provide early warning of such attacks, so that appropriate countermeasures may be taken to combat them.

Quick Links

- What's New?
- Argos Secure S...
- NoAH security i...
- Survey of secu...

Latest News

- 17 May '06 [The 1st NoAH workshop organised at the TNC 2006 conference in Catania, Italy.](#)
- 2 May '06 [Network-level Polymorphic Shellcode Detection using Emulation](#)
- 19 Apr '06 [NoAH to organise workshop on honeypots at TNC 2006](#)
- 4 Apr '06 [Distributed Honeypot Project survey reinforces NoAH conclusions](#)
- 28 Mar '06 [Argos secure system emulator source code available](#)
- 17 Mar '06 [NoAH survey on honeypot requirements summarised](#)
- 17 Mar '06 [Use of Shadow Honeypots presentation available](#)

First NoAH Work...

The NoAH project c...
TNC 2006 conferen...
held during the wo...

Mailing List

Please join the No...
project developme...

Disclaimer

The NoAH Project...
any opinions expre...
reflect the official...
Contract No. RIDS...

Argos: An Emulator for Capturing Zero-Day Attacks - Firefox

File Edit View Go Bookmarks Tools Help

http://www.few.vu.nl/argos/

ARGOS

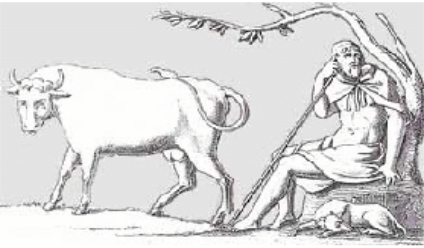
Home Features Downloads News Documentation Contribute Links Contact Search

What is Argos?


Argos is a full and secure system emulator designed for use in honeypots. It is based on [Qemu](#), an open source emulator that uses dynamic translation to achieve a fairly good emulation speed.

Argos extends Qemu to enable it to detect remote attempts to compromise the emulated guest operating system. Using dynamic taint analysis it tracks network data throughout execution and detects any attempts to use them in an illegal way. When an attack is detected the memory footprint of the attack is logged.


Argos is the first step to create a framework that will use next generation honeypots to automatically identify and produce remedies for zero-day worms, and other similar attacks. Next generation honeypots should not require that the honeypot's IP address remains un-advertised. On the contrary, it should attempt to publicise its service and even actively generate traffic. In format honeypots this was often impossible, because malevolent and benevolent traffic could not be distinguished. Since Argos is explicitly signalling each possibly successful exploit attempt, we are now able to differentiate malicious from innocuous traffic.




GET ARGOS



Download Argos





Today 011/0014

http://www.fp6-noah.org

News & Events

22/06/2006
Argos developers mailing list available.
A developers list for Argos is available at last. To post to the list you will have to subscribe [here](#). You can also browse through the [mailing list's archives](#).
[Read more →](#)

13/06/2006
Argos release 0.1.4
A new version of Argos has been released.

NoAH partners

- ~ Foundation of Research and Technology (FORTH), Heraklion, Greece – coordinator
- ~ Vrije Universiteit, Amsterdam, The Netherlands
- ~ ETH, Zurich, Switzerland
- ~ TERENA, Amsterdam, The Netherlands
- ~ FORTHnet SA, Heraklion, Greece
- ~ DFN-CERT, Hamburg, Germany
- ~ Virtual Trip Limited, Greece
- ~ ALCATEL, France

Backup slides

Funneling



- ~ arpd to collect IP addresses
 - ~ user-space daemon that responds to ARP requests arriving to the network interface of the honeypot
- ~ honeyd handles traffic arriving at honeypots
- ~ funneling has no overhead
 - ~ we tested emulating /24, /16, and /8 subnets without any noticeable difference in performance

Tunneling

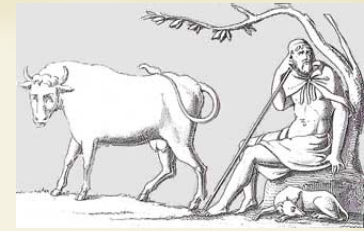


- ~ OpenVPN 2.0 as tunnel software
- ~ Encrypted channel, supports packet compression

Honey@home - challenges

- ~ We cannot trust clients
 - ~ Anyone will be able to set up honey@home
- ~ Clients must not know the address of honeypot
 - ~ Honeypots may become victims of flooding
- ~ Address of client must also remain hidden
 - ~ Attacker can use their black space for flooding
- ~ Computer-based mass installation of honey@home mockup client should be prevented

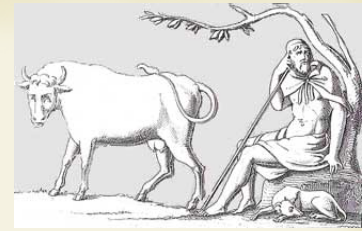
Network Data Tracking



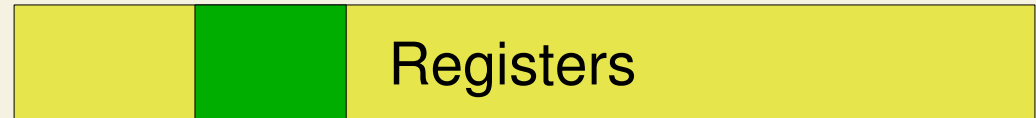
Reg B = network_read

Registers

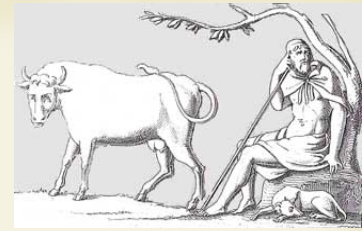
Network Data Tracking



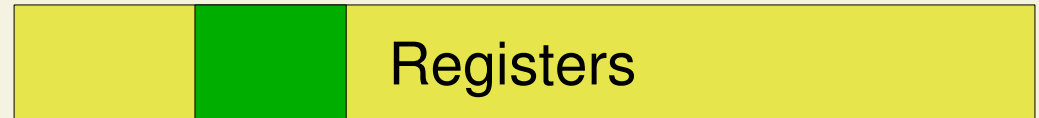
Reg B = network_read



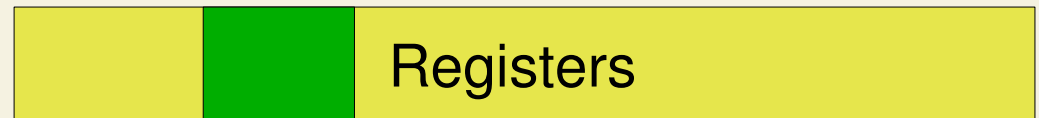
Network Data Tracking



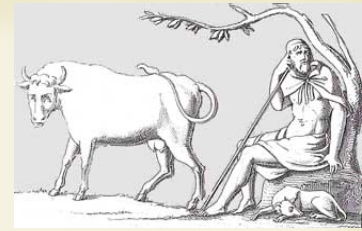
Reg B = network_read



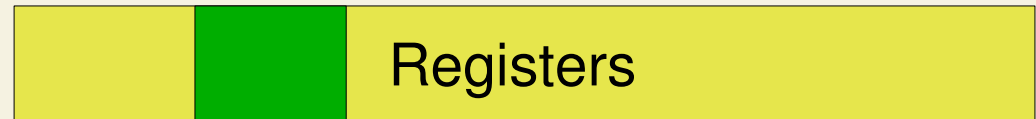
Reg A = Reg A + Reg B



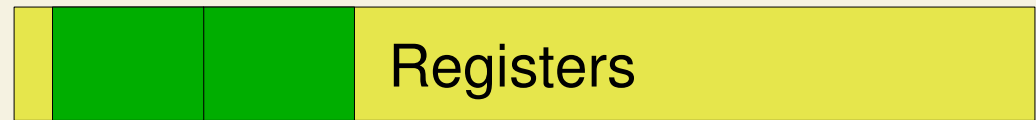
Network Data Tracking



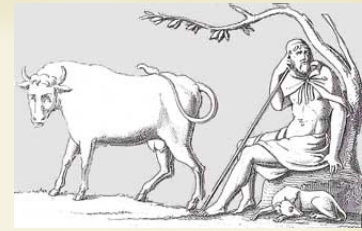
Reg B = network_read



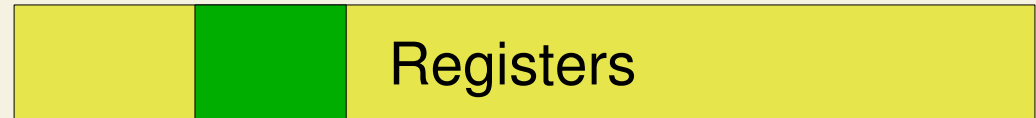
Reg A = Reg A + Reg B



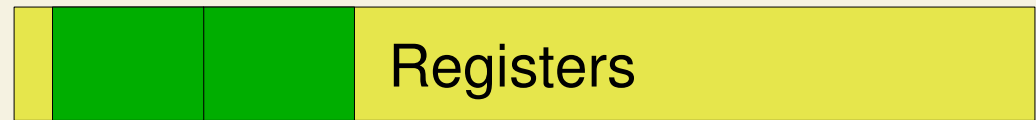
Network Data Tracking



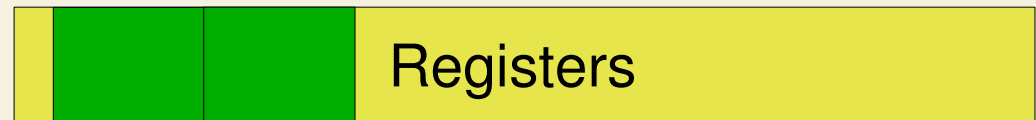
Reg B = network_read



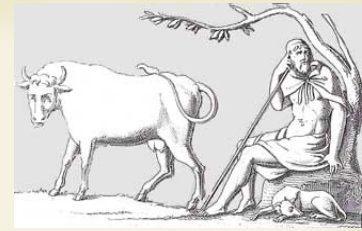
Reg A = Reg A + Reg B



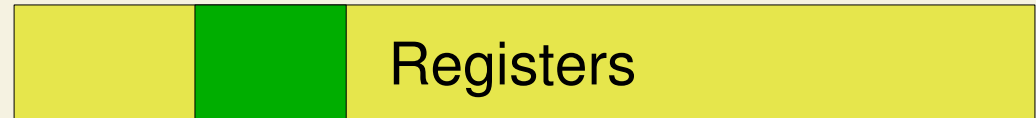
Memory(B) = Reg B



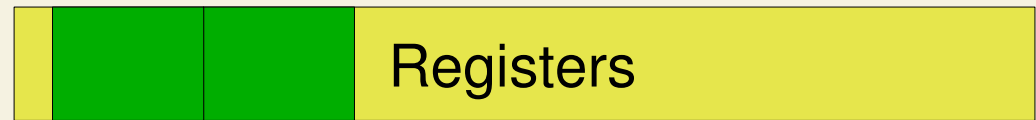
Network Data Tracking



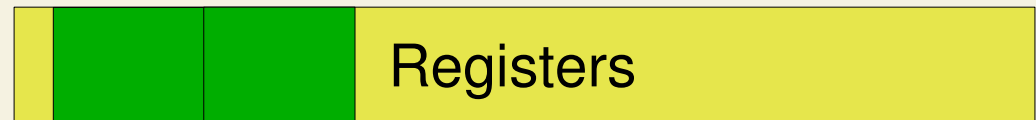
Reg B = network_read



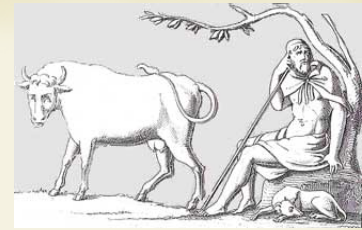
Reg A = Reg A + Reg B



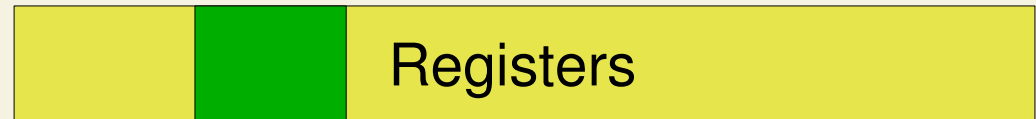
Memory(B) = Reg B



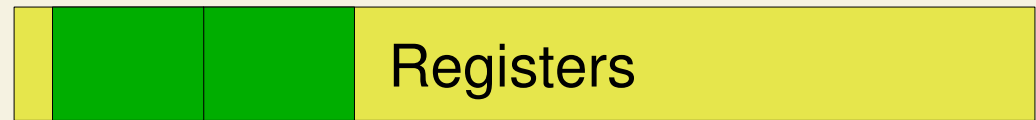
Network Data Tracking



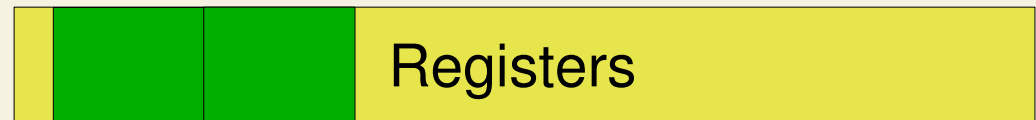
Reg B = network_read



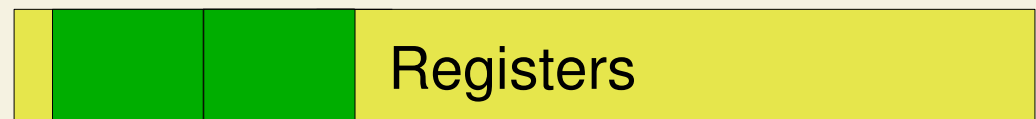
Reg A = Reg A + Reg B



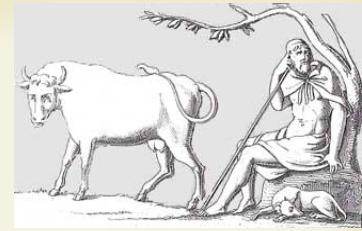
Memory(B) = Reg B



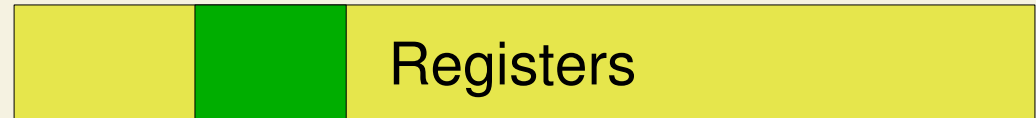
Reg B = Reg A / 12.34
(Sanitise data)



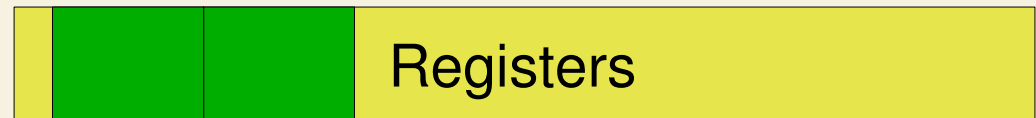
Network Data Tracking



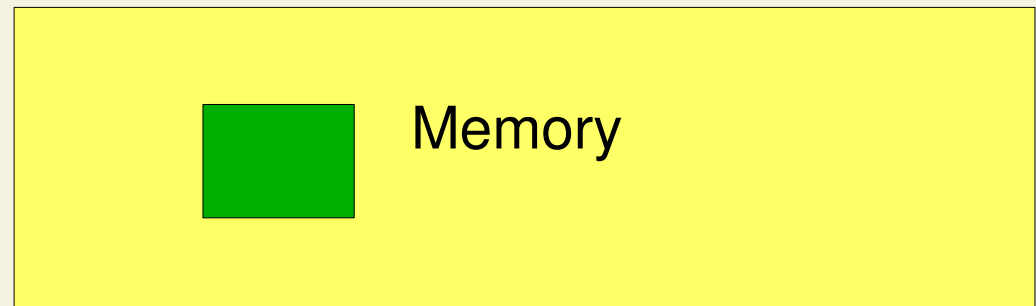
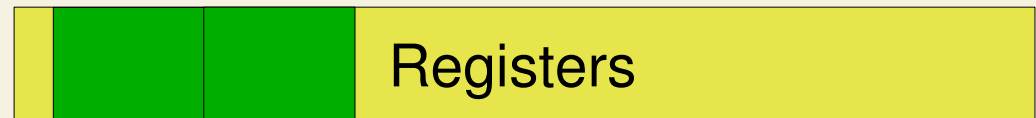
Reg B = network_read



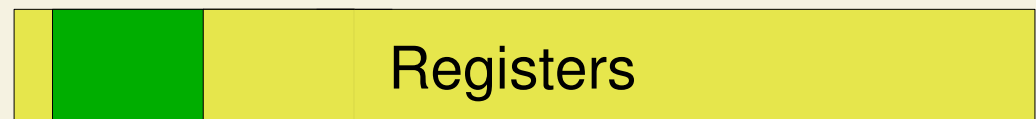
Reg A = Reg A + Reg B



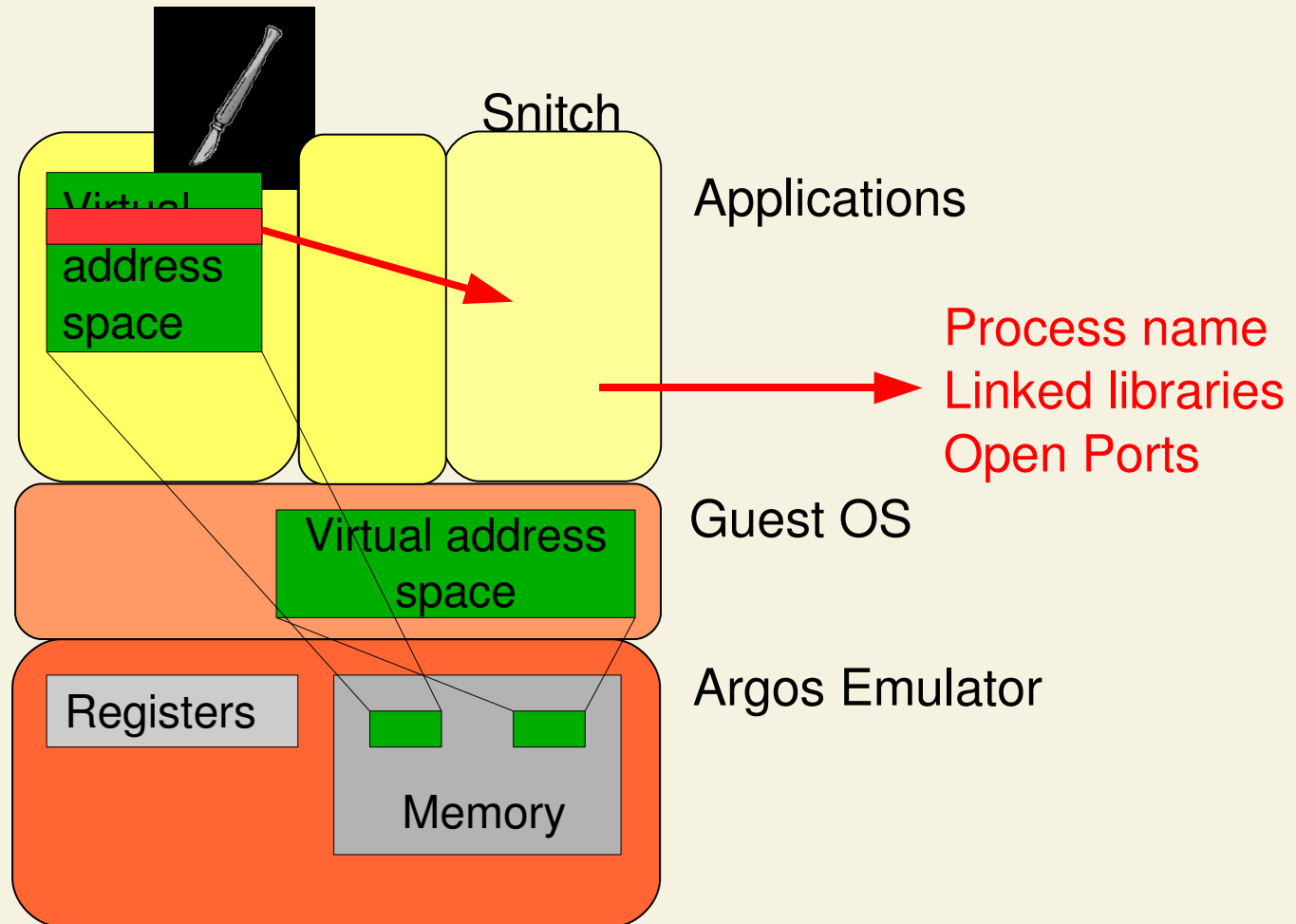
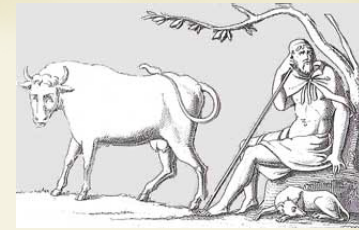
Memory(B) = Reg B



Reg B = Reg A / 12.34
(Sanitise data)

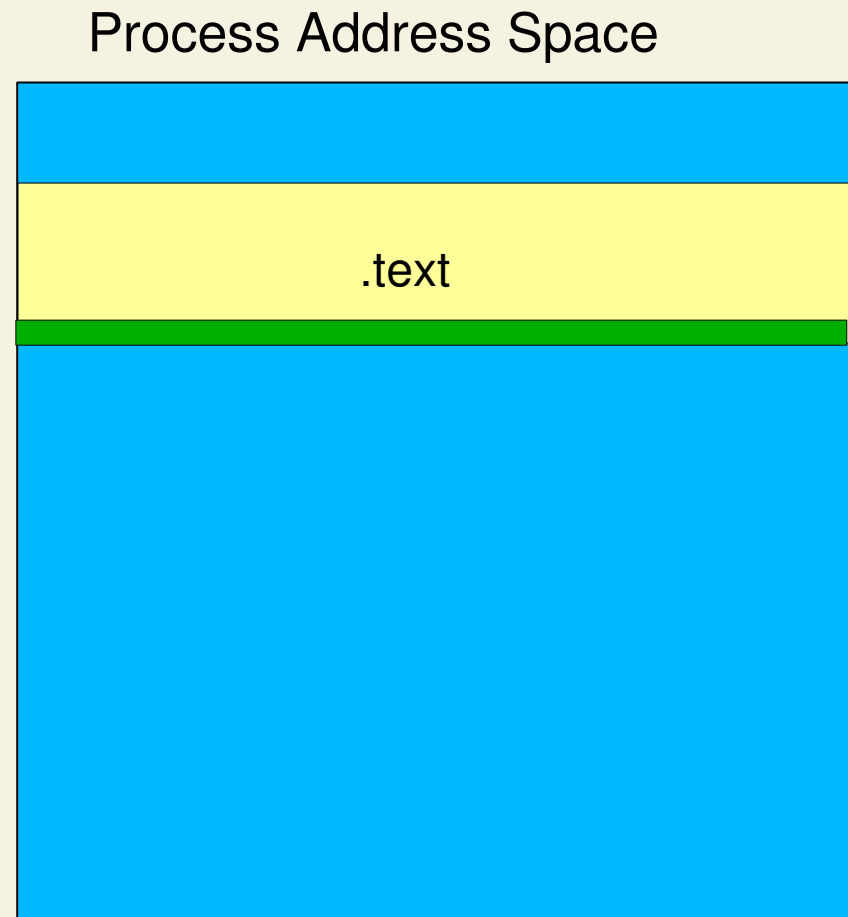


Guest forensics



Forensics shellcode injection

- ~ lookup process's read-only pages
- ~ inject code at last text segment page
- ~ point EIP to shellcode



Forensics

- ~ pid = getpid()
- ~ connect(localhost)
- ~ send(pid)

Snitch

- ~ listen()
- ~ accept()
- ~ read(pid)
- ~ exec(netstat or
OpenPorts)
- ~ connect(argos host)
- ~ send(info)

Network tracking

