Adapting applications to exploit virtualization management knowledge

Vitalian A. Danciu and Alexander Knapp

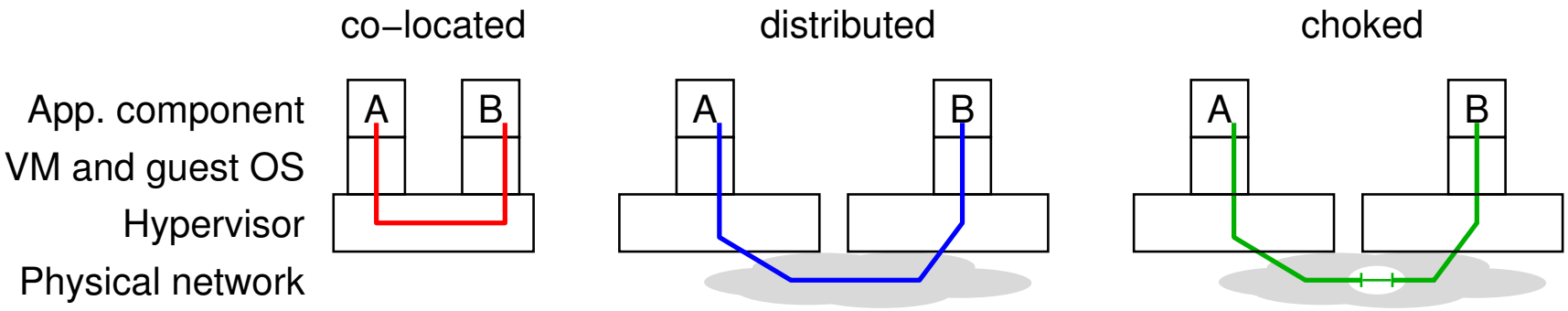
DMTF SVM 2013

MNM TEAM

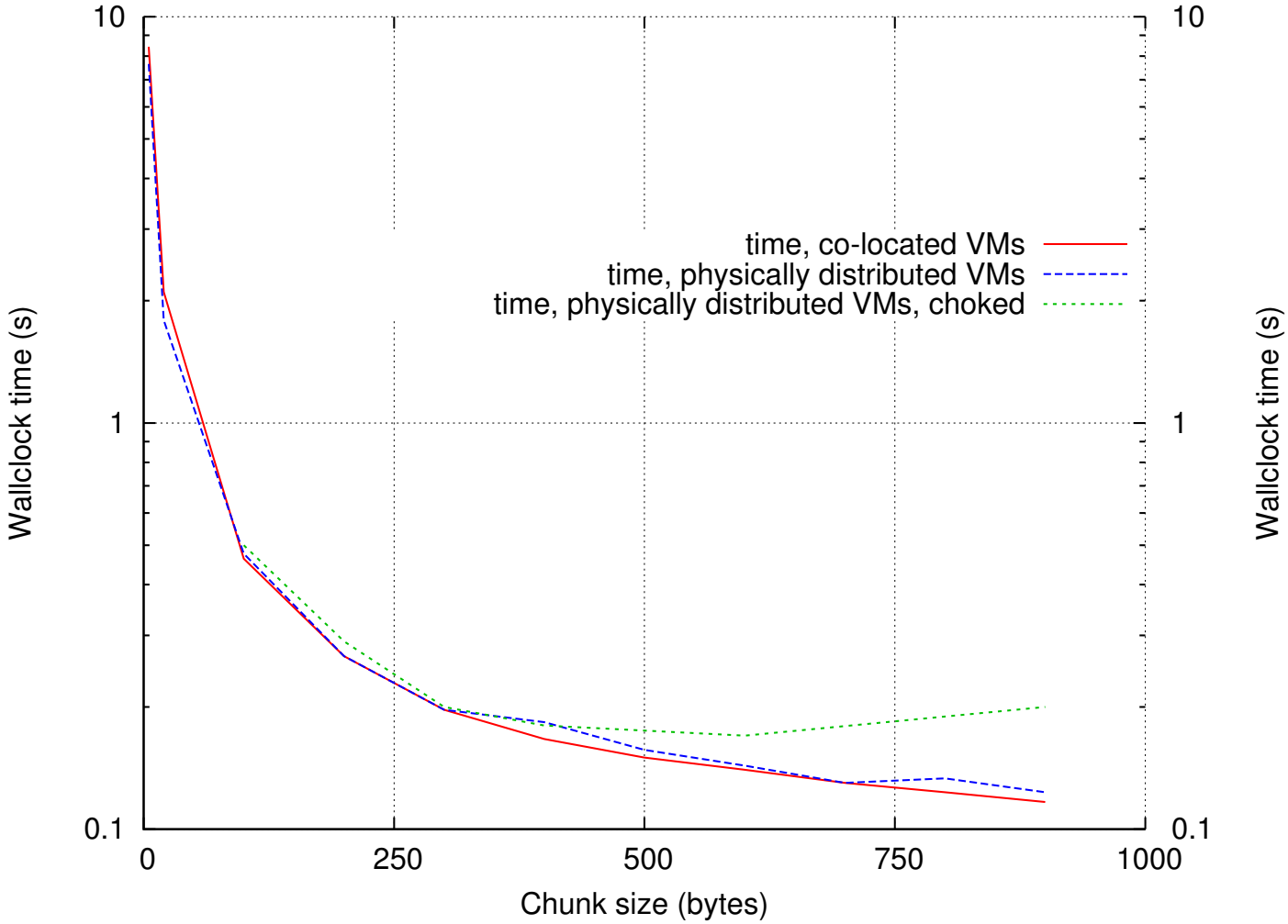
Applications running on virtualized infrastructure suffer!

1. Example of “suffering”, by experiment
2. How to adapt applications’ behaviour
3. Relation between application and the management system

An Inter-VM communication experiment

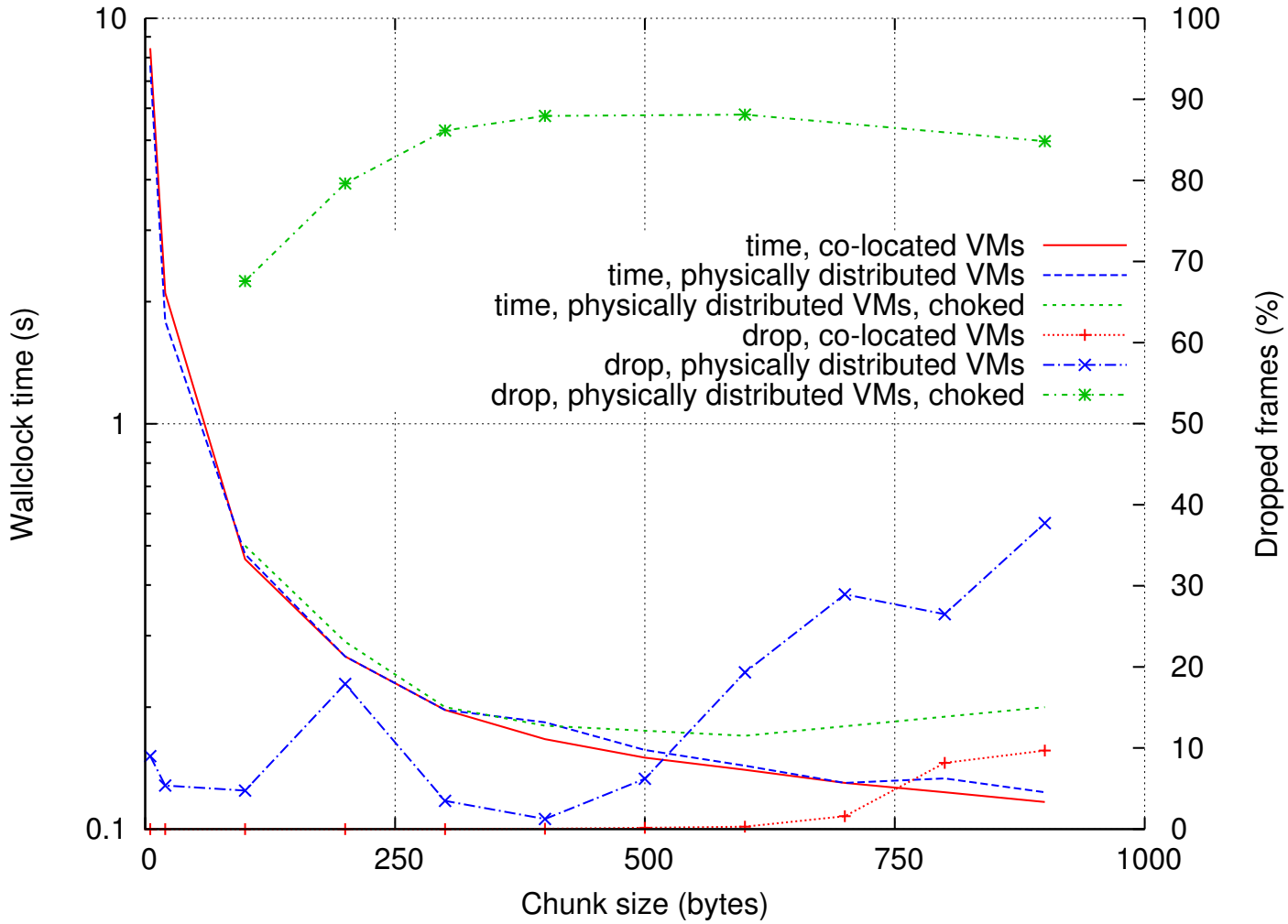


Transmission time for 10^7 bytes UDP payload (sender's view)



Performance seems equal; how can this be?

Segment drop rate (receiver's view)



Deployment setup can change at any time: how can applications adapt?

Problem

Observation: Deployment changes quickly, but application behaviour does not.

- Application operates from static-world assumptions.
- Virtualization masks deployment state from application

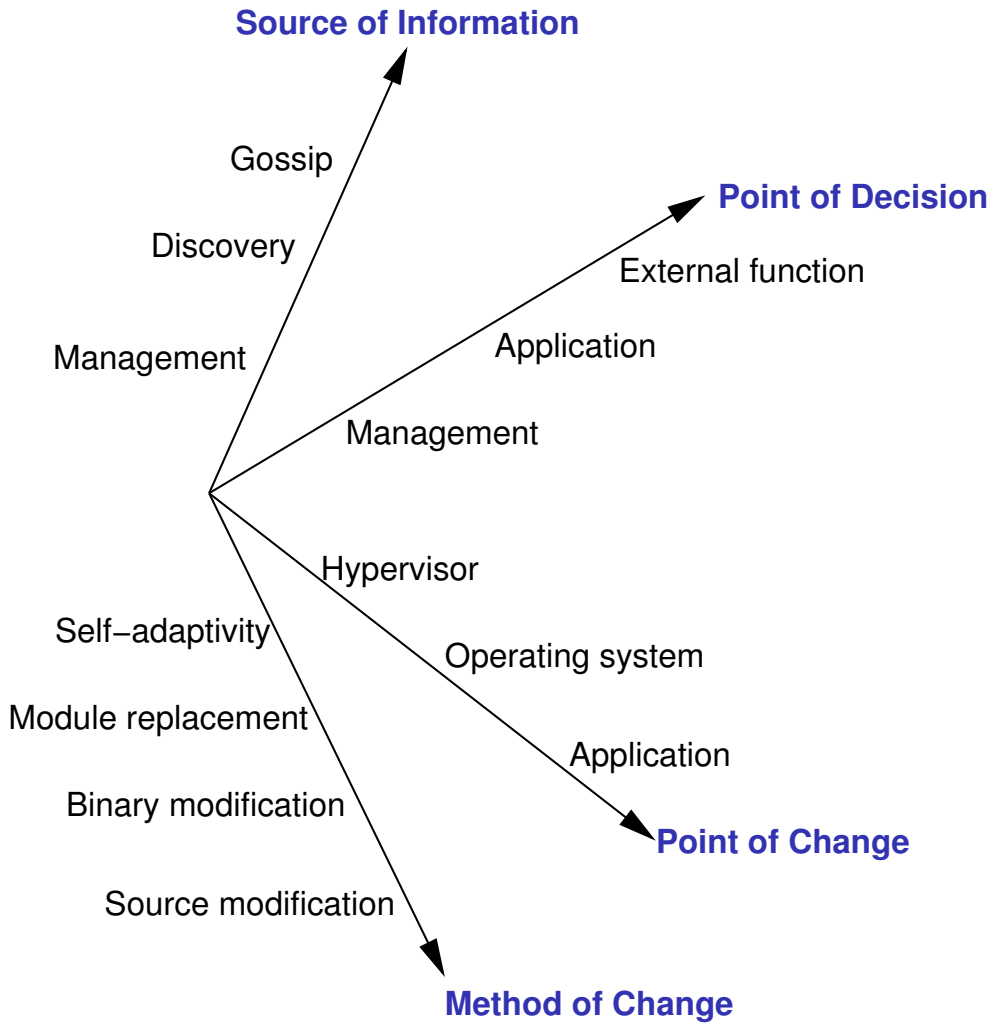
$\Rightarrow$ Detrimental effects in some of the states

Scope of the problem (network thoughput is but an example!)

- locality: communication metrics (throughput, delay, faults, ...)
- resources: CPU capacity, RAM, ...
- context: security, hardware capabilities, ...

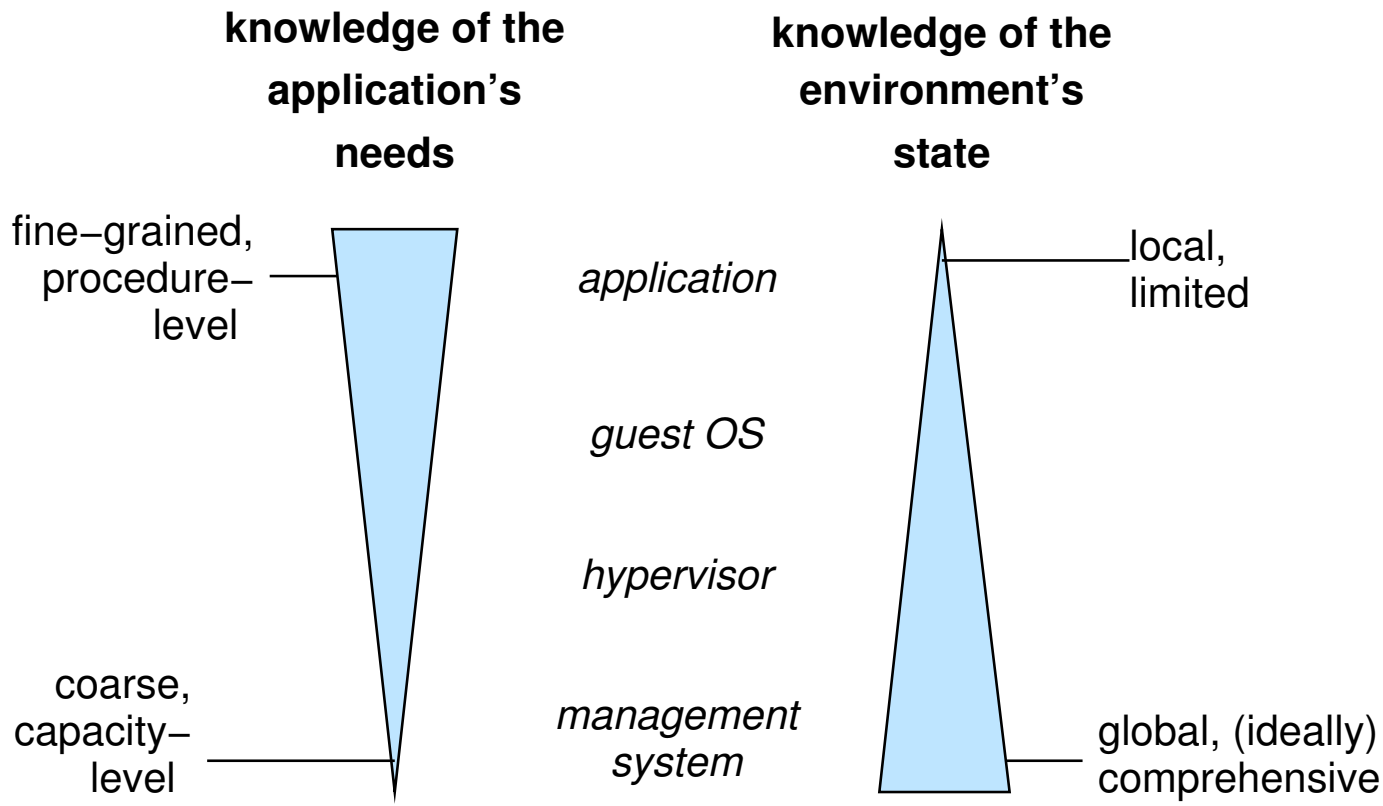
How to render application software virtualization-aware?

Solution dimensions



Select one point in this solution space!

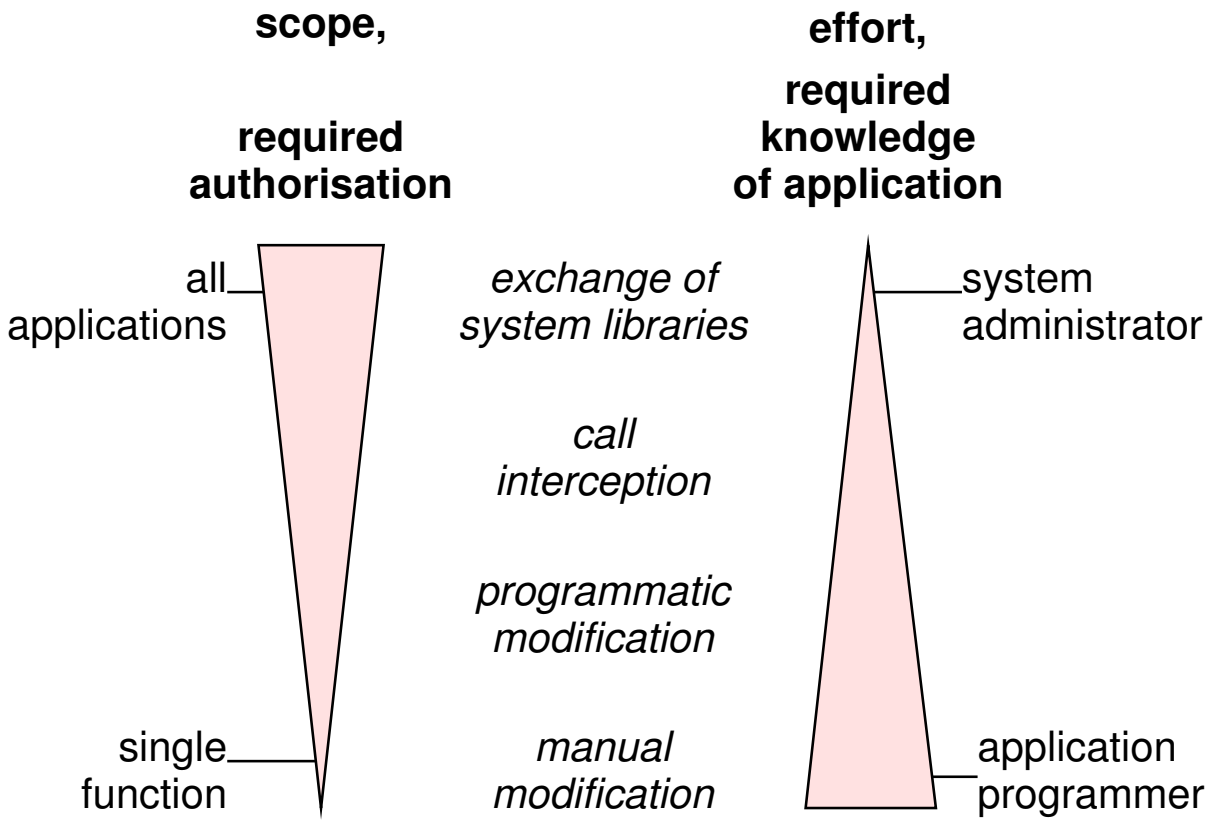
Source of information, point of decision



Choices

- Management system to provide environmental information/guidance
- Application to decide on it

Point and method of change



Choices

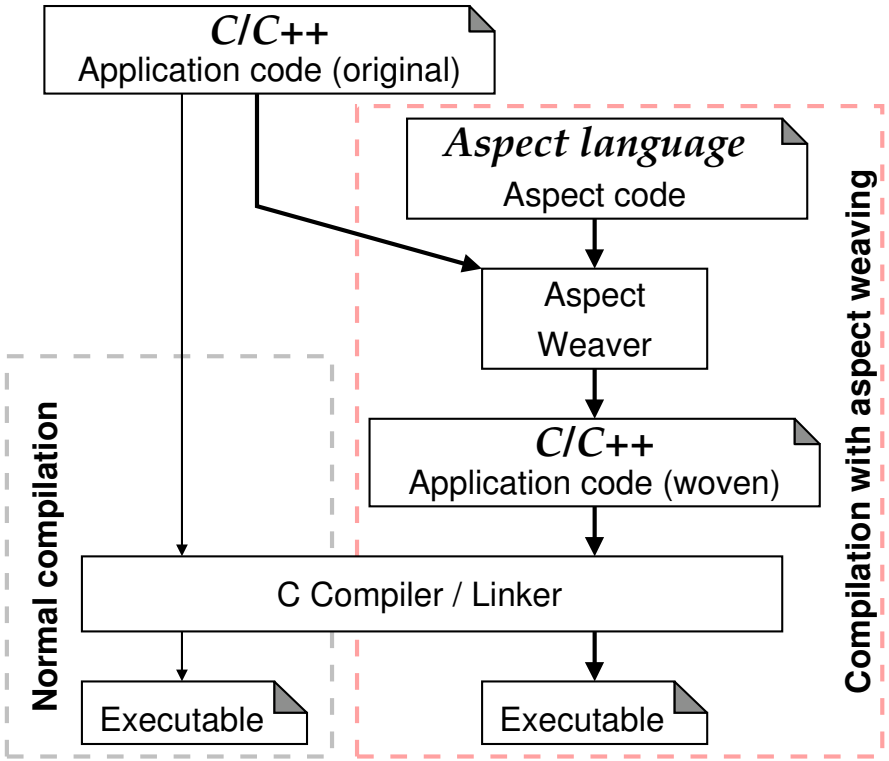
- Modification of application code (source or binary)
- Programmatic, machine-supported modification

Code example: output to a UDP socket


```
1  int write_udp(const char* targetip,
                 unsigned int port,
                 long count) {
     int transmit_socket = socket(AF_INET,
5                                 SOCK_DGRAM,
                                  IPPROTO_UDP);
     connect(transmit_socket,
             (struct sockaddr*)&si_other,
             sizeof(si_other));
10   int c = 0;
     while (c++ < count)
       write(transmit_socket,
13           (void*)chunk, (size_t)sbuf);
   }
```

Aspect-oriented programming (AoP) at a glance

"Glorified string substitution with knowledge of the programming language"



- Aspect: "a cross-cutting concern"
- Aspect language
  - matching/scoping of language constructs (classes, methods, . . .)
  - access to program structures
  - manipulation of parameters, return values, . . .
- Pointcut expressions: where to modify
- Advice expression: how to modify
- Weaving: integration of Aspect code

Example: a buffering aspect (for both sockets and files)


```
1  aspect Buffering {
     pointcut openpc() =  "% ...:: open(...)";
3    pointcut socketpc() = "% ...:: socket(...)";
4    advice call(openpc()||socketpc()) : around() {
5      tjp->proceed();
6      int myfh = *((int*)tjp->result());
7      writebuf[myfh] = (char*)malloc(bufsize);
8    }
     pointcut writepc() = "% ...:: write(...)";
10   advice call(writepc()) : around() {
       int fd = *((int*)tjp->arg(0));
       const void* buf = *((const void**)tjp->arg(1));
       unsigned int count = *((unsigned int*)tjp->arg(2));
       int myfh = _getmyfh(fd);
15     if ((buffill[myfh] + count) < bufsize) {
16       memcpy(writebuf[myfh] + buffill[myfh], buf, count);
         buffill[myfh] += count;
       }
       else {
20       write(fd, writebuf[myfh], buffill[myfh]);
21       buffill[myfh] = 0;
         tjp->proceed();
23     }
       *((int*)tjp->result()) = count;
25   }
   }
```

← Intercept the function calls, that create file handles.

← Allow the call to execute

← and to yield a file handle,

← Associate the file handle with a buffer.

← Intercept the `write()` calls.

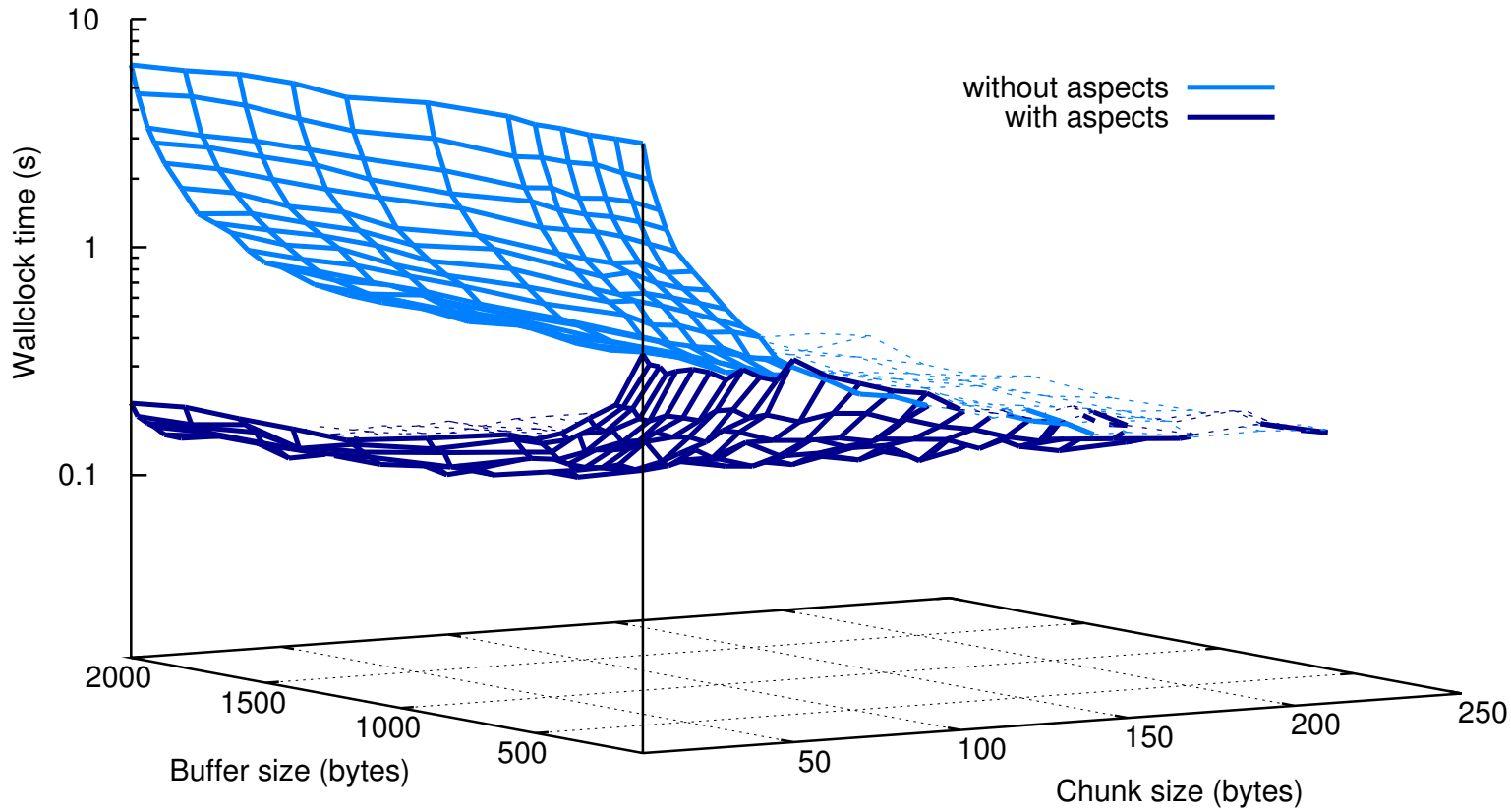
← Write to buffer, if room

← When full, flush buffered data,

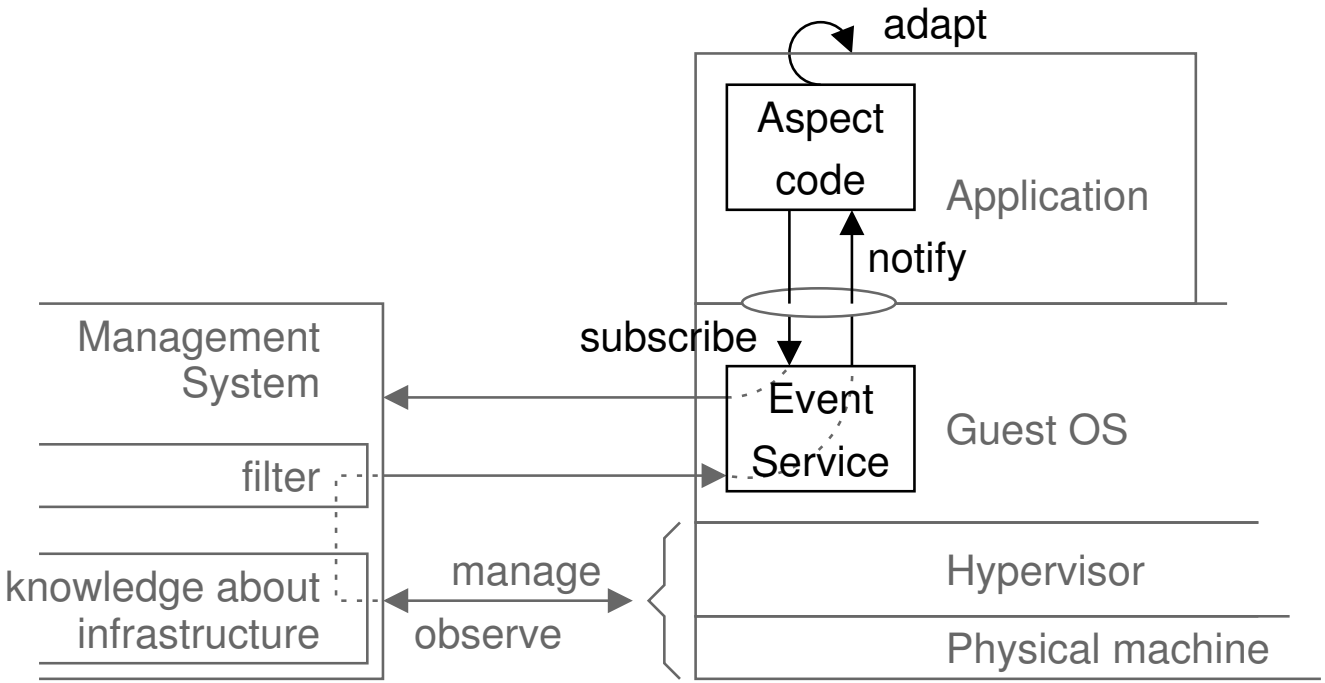
← and allow the current `write()` to execute.

← Always return the expected value.

Behaviour of original vs. woven code



Supplying management information to applications



Conclusions and Open questions

Your software runs on virtualized infrastructure —and it suffers!

- Environment has changed; application code has not.
- Need to adapt application code, but
  - not manually (too large code base)
  - not centrally (different applications have different needs)
  - not to be self-adaptive (selfish adaptation obviates management goals)

Our AoP approach works, however it has limitations

- Applicability determined by code quality → quality metrics?
- Conflicting aspects→ balancing? aspect “patterns”?

Management is capable to supply global information, however

- it may not wish to (public XaaS scenarios) → discovery, gossip?
- which information is relevant to ask? → situation/cause/effect? formalism?