

# Frontiers of Software Maintenance: Program Slicing

Dave Binkley  
Loyola College

Keith Gallagher  
Durham University

# The Outline Slide!

- 
- 
- 
- 
- 
- 
- 
-

# The Little Slicer: Definition

x = 42

d = 23

d

x = 42 // !!

d = 23

# The Little Slicer: Data Dependence

$a = 42$

$x = 2$

$b = 23 + a$

$y = 3$

$c = b + 2$

$c$

$a = 42$

$x = 2$

$b = 23 + a$

$y = 3$

$c = b + 2$

# The Little Slicer: Control Dependence

```
a = 42
while (B)
{
    a = b - 2
    x = 10
    b = 64
}
c = a + 2
```

c

```
a = 42
while (B)
{
    a = b - 2
    b = 64
}
c = a + 2
```

# The Little Slicer: Meaningful Semantics

```
sum = 0
prod = 1
i = 1
while ( i < 11)
{
    sum = sum + i
    prod = prod * i
    i = i + 1
}
```

prod

```
sum = 0
prod = 1
i = 1
while ( i < 11)
{
    prod = prod * i
    i = i + 1
}
```

# The Little Slicer: hmmm....

```
while ( true)
```

```
{
```

```
    x = 1
```

```
}
```

```
y = 2
```

*y*

*y = 2*

# The Little Slicer: OOOWWW!!!

```
if (b)  
    goto L2
```

```
L1:  
    y = 1  
    goto L3  
    z = 2
```

```
L2:  
    x = 3  
    goto L1
```

```
L3:  
print x  
print y
```

```
L1:  
    y = 1  
    goto L3
```

```
L3:  
print y
```

y

# The Little Slicer

## Other Techniques: Forward

```
sum = 0
```

```
prod = 1
```

```
prod prod = 1
```

```
i = 0
```

```
while( i < 11)
```

```
{
```

```
    sum = sum + i
```

```
    prod = prod * i
```

```
prod = prod * i
```

```
    i = i + 1
```

```
}
```

# The Little Slicer

## Other Techniques: Dynamic

```
// input 42
```

```
// input 42
```

```
read (a)
```

```
if ( a < 0)
```

```
    a = -a
```

```
x = 1 / a
```

```
x
```

```
read (a)
```

```
x = 1 / a
```

# The Little Slicer

## Other Techniques: Conditioned

```
// a > 0
```

```
// a > 0
```

```
read (a)
```

```
read (a)
```

```
if ( a < 0)
```

```
    a = -a
```

```
x = 1 / a
```

```
x
```

```
x = 1 / a
```

# The Little Slicer

## Other Techniques: Amorphous

```
sum = 0
```

```
prod = 1
```

```
i = 0
```

```
while( i < 11)
```

```
{
```

```
    sum = sum + i
```

```
    i = i + 1
```

```
}
```

sum

sum = 55

# The Little Slicer

## Applications: Dicing

```
main() {  
    int c, nl, nc;  
  
    nl = 0;  
    nc = 1;  
    c = getchar();  
    while ( c != EOF ) {  
        nc = nc + 1;  
        if ( c == '\n' ) nl = nl + 1;  
        c = getchar();  
    }  
    printf("%d \n", nl);  
    printf("%d \n", nc);  
}
```

# The Little Slicer

## Applications: Regression Testing

```
main() {  
    int c, nl, nc;  
    nl = 1;  
    nc = 0;  
    c = getchar();  
    while ( c != EOF ) {  
        nc = nc + 1;  
        if ( c == '\n' ) nl = nl + 1;  
        c = getchar();  
    }  
    printf("%d \n", nl);  
    printf("%d \n", nc);  
}
```

# The Little Slicer

## Applications: Maintenance

```
inword = NO;
nl = 0;
nw = 0;
nc = 0;
c = getchar();
while ( c != EOF ) {
    nc = nc + 1;
    if ( c == '\n' ) nl = nl + 1;
    if ( c == ' ' || c == '\n' || c == '\t' )
        inword = NO;
    else if ( inword == NO )
        { inword = YES;
          nw = nw + 1; }
    c = getchar();
}
```

Challenges Trends Future

# Challenge 1

Programs are less static

Harder to statically predict code used  
(an issue for all static analysis)

Consider

imperative  $\rightarrow$  OO  $\rightarrow$  agents

# Challenge 2

## *Observation*

slicing has failed  
to capture the  
***intuition*** that  
engineers bring  
to programming

# Challenges - Intuition

*Observation*

slicing has failed to capture the intuition

....

This means  
opportunity!!

For you!

# Challenges - Intuition

For example one might ask

**Why** has slicing failed to capture the intuition ....

At an average  $1/3$  of the program,  
perhaps slices are too large

# Challenges - intuition

One recent proposal ***thin slicing***

A thin slice includes only  
***producer statements***

|               |                          |
|---------------|--------------------------|
| a = new A()   | // produces object o1    |
| a.f = new F() | // produces o1.f         |
| d = a         | // not a producer        |
| x = d.f       | // uses o1.f (but not d) |

# From Challenges to Trends

But first the dream

# The (original) Dream

Now I'll just  
*slice* on sum

Programmer  
in Pink



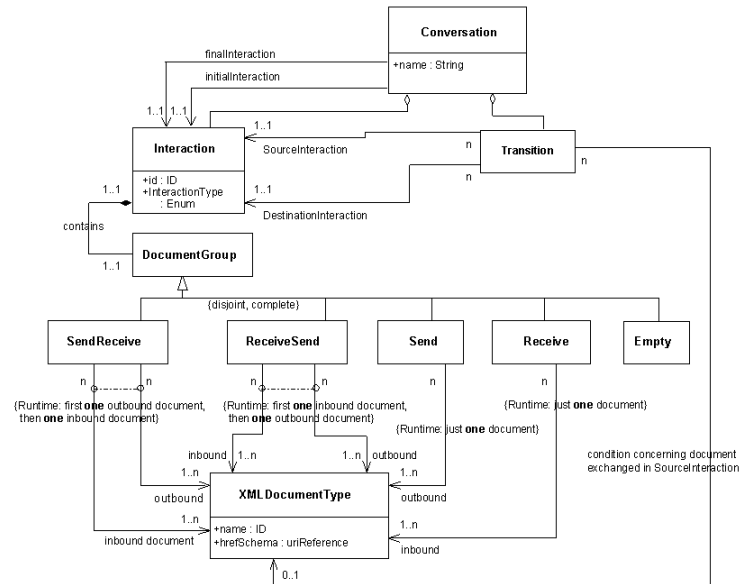
# Current Trends (the Reality)

Slicing used to *aid other analyses*

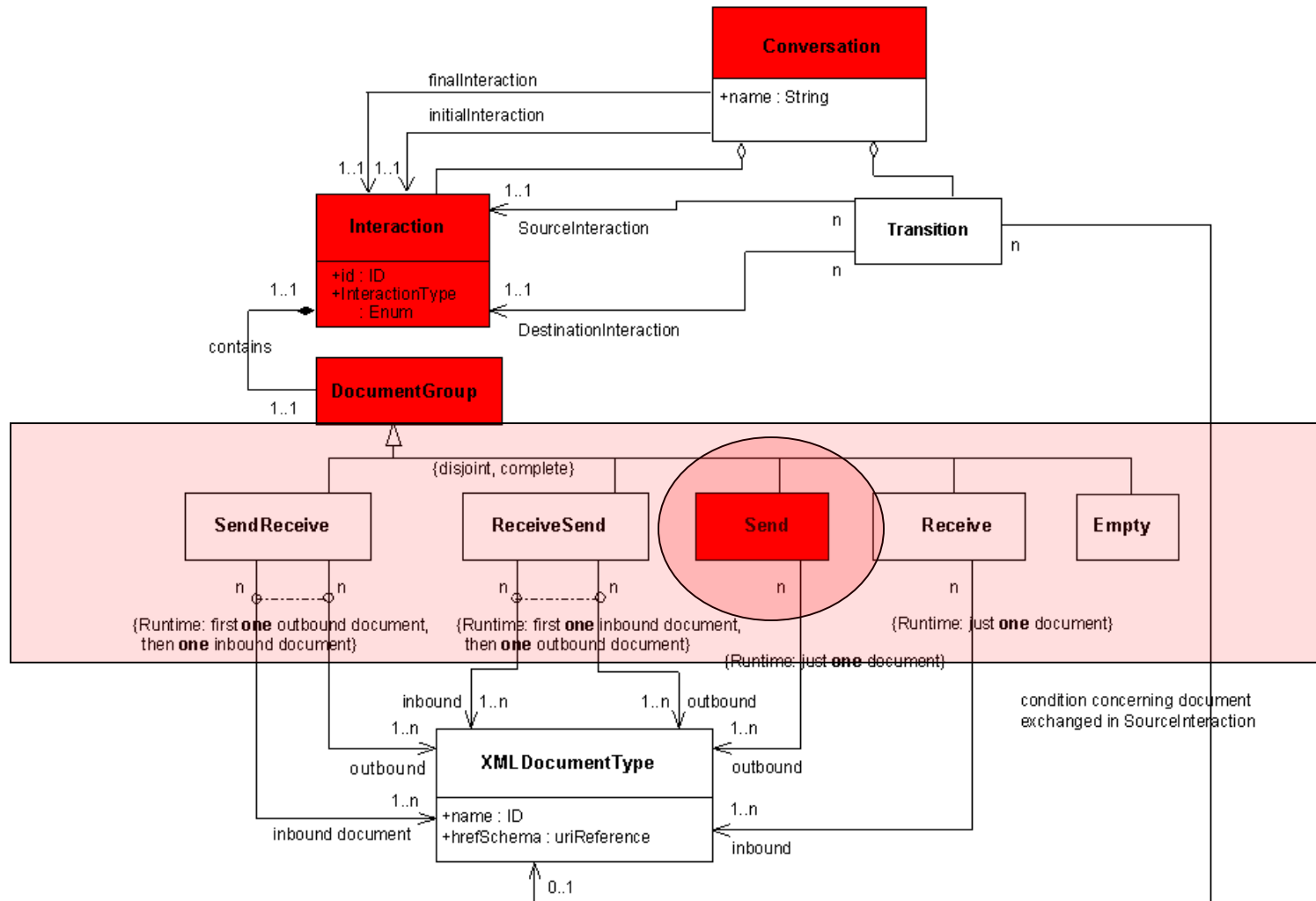
- in part
  - e.g., Paul's SCAM Codesurfer Keynote
- in whole
  - model checking
  - dependence structure
- in theory

# Consider model checking when verifying a property

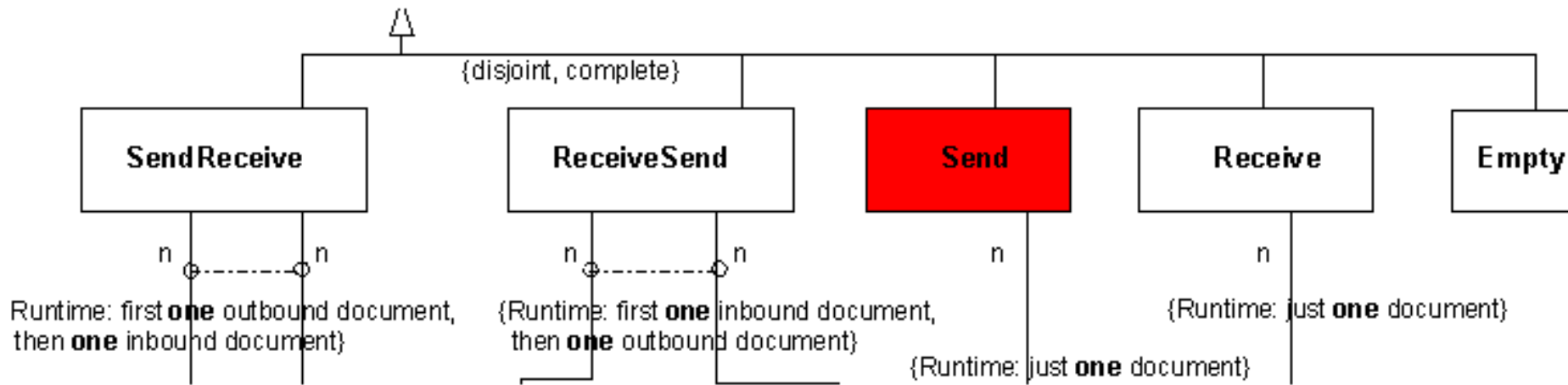
# Is there a *memory leak* when sending a message?



# Slicing as an Aid



# Slicing as an Aid

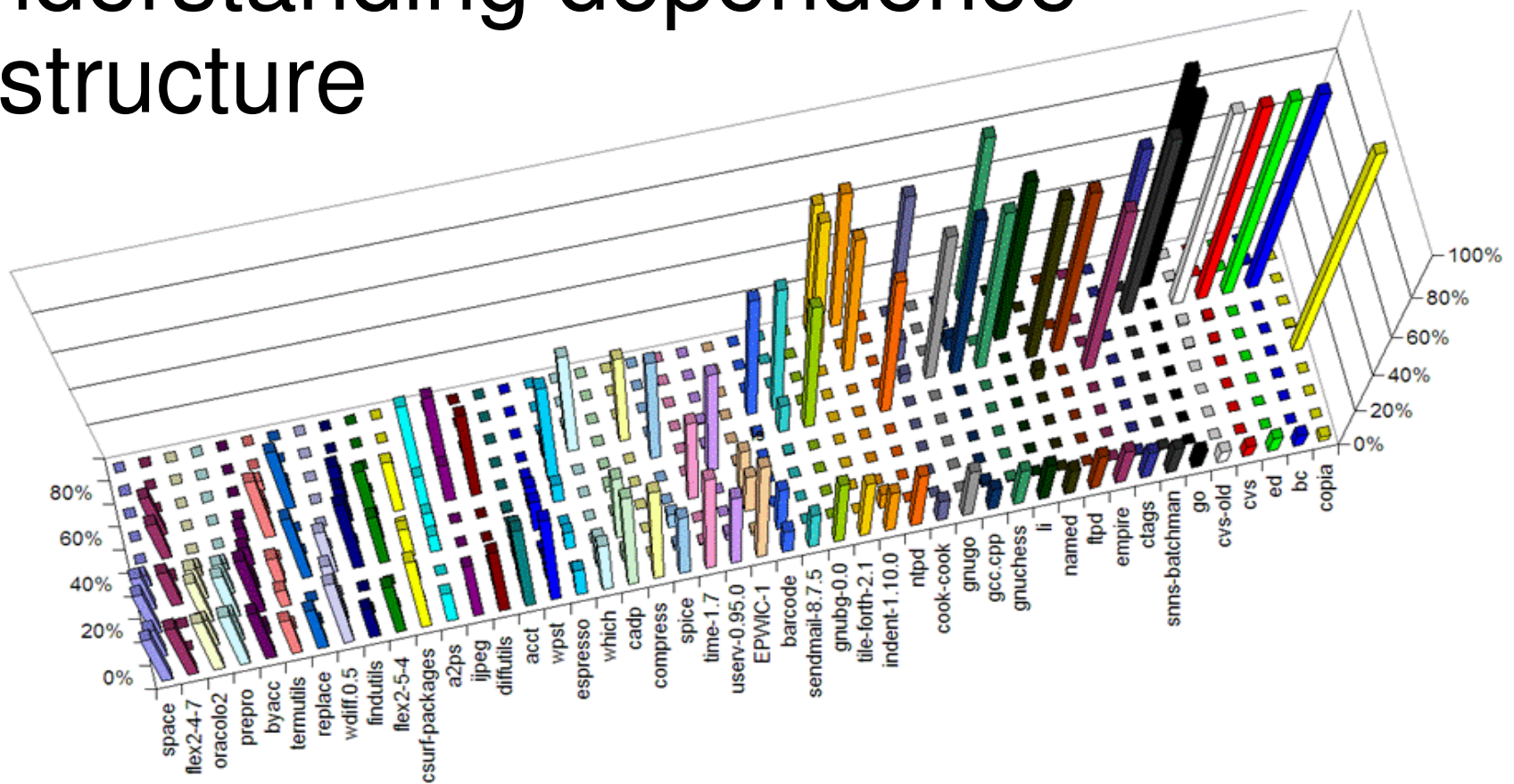


“5” times faster after slicing!

# Trend 2

## Slicing as an Aid

Understanding dependence structure



# (commitment) Dependence

I'll go if you go



# Dependence Cluster

I'll go if you go

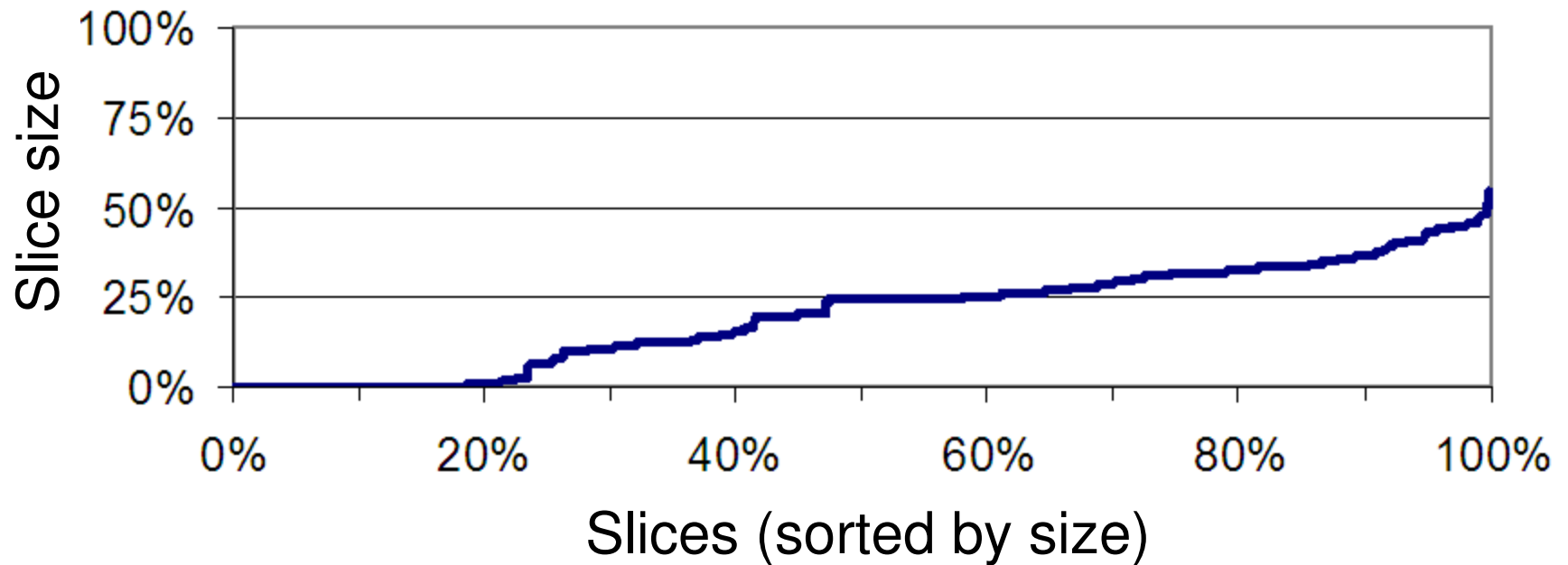
I'll go if you go

I'll go if you go



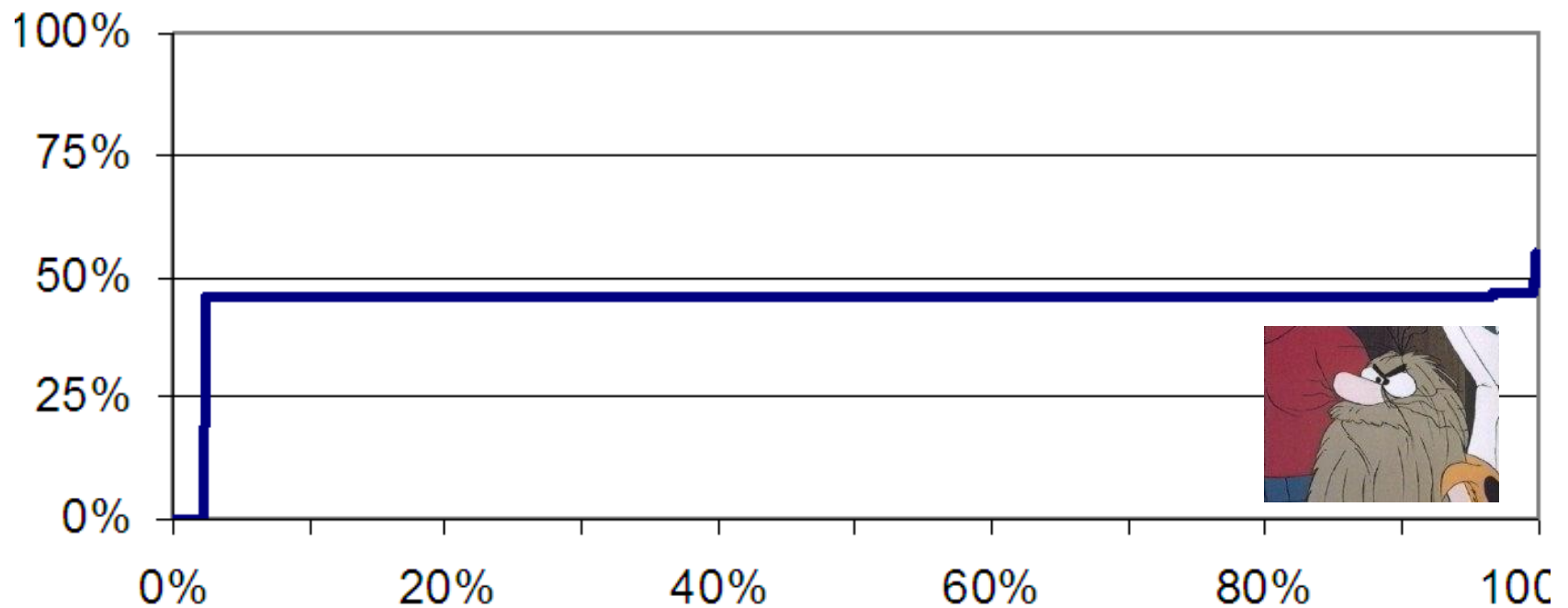
Thus you get  
all or nothing!

# Dependence Structure



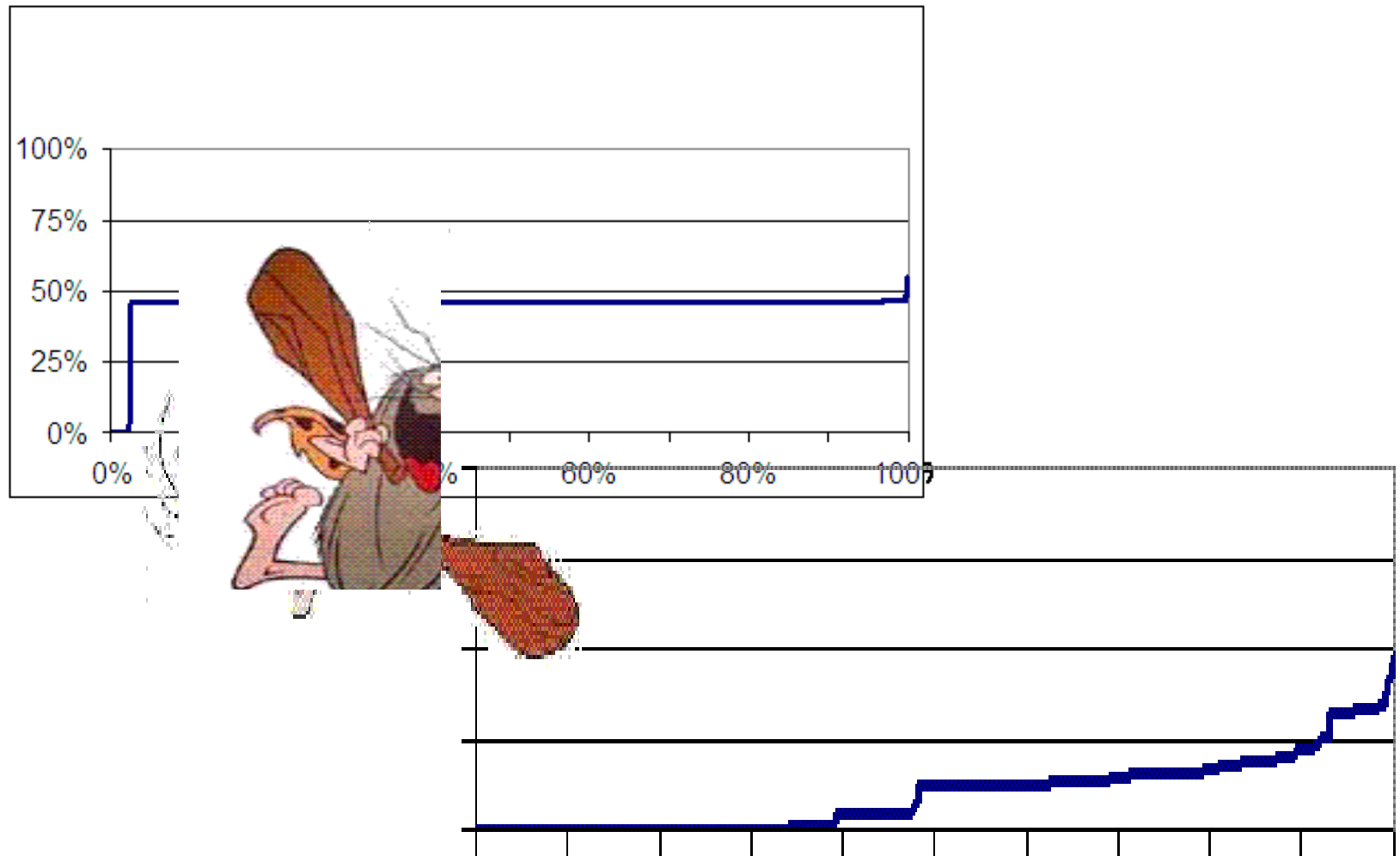
# Dependence Structure

## A Second Program

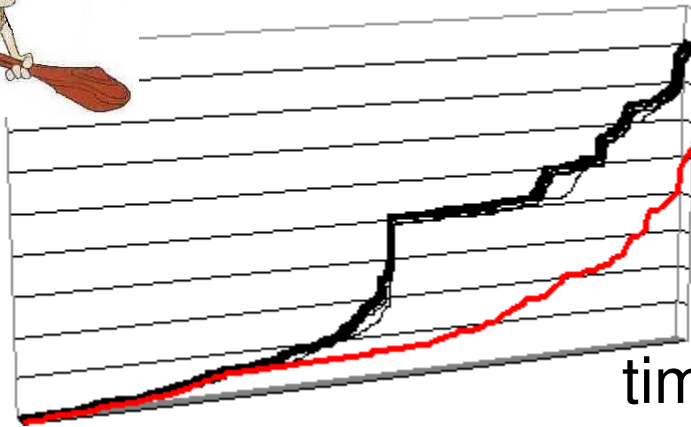


# Dependence Structure

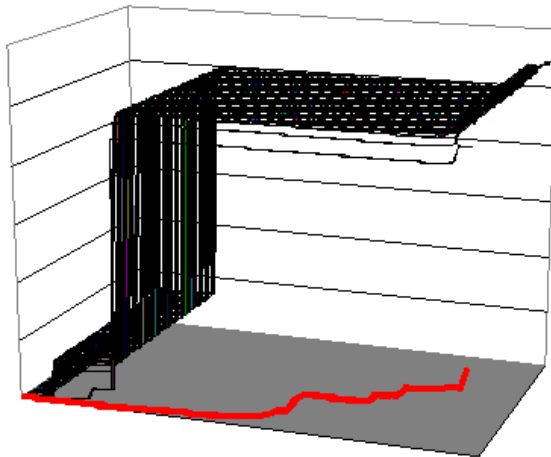
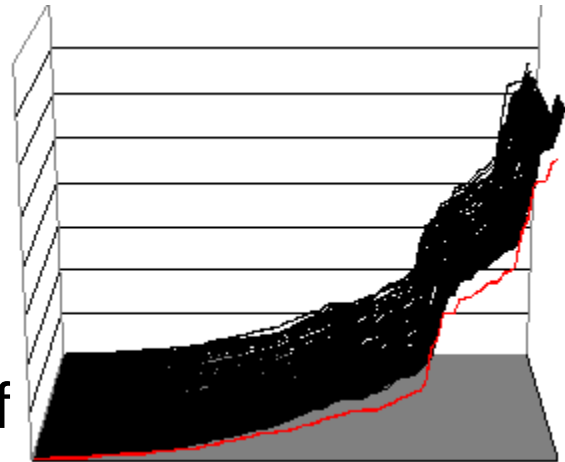
## Breaking Dependence Clusters



# Dependence Structure Automated Breaking

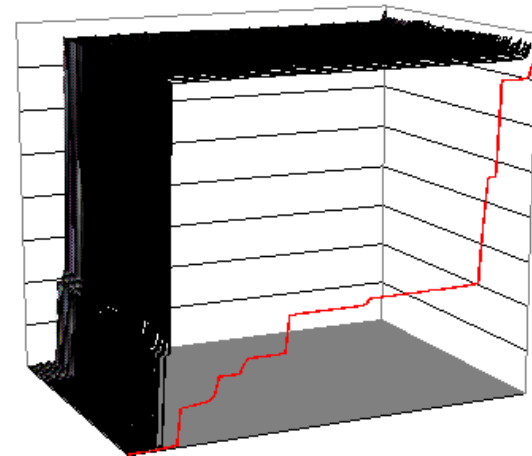


time wdiff



ctags

barcode



# Trend 3 In Theory

Better Formal Understanding



# Formalization

Relate traditional  
***operational*** semantics to  
more declarative  
***denotational*** semantics

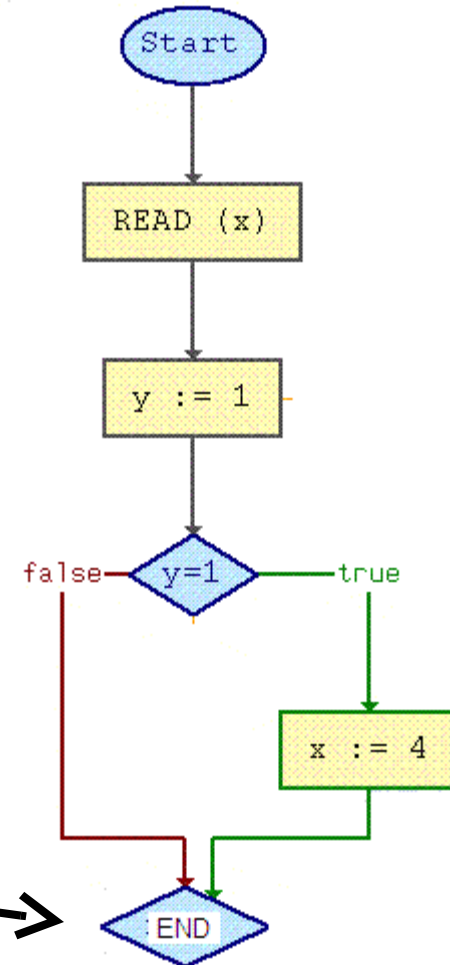
# Formalization

Semantics of *reactive systems*

Example - rethink of  
control dependence  
definition

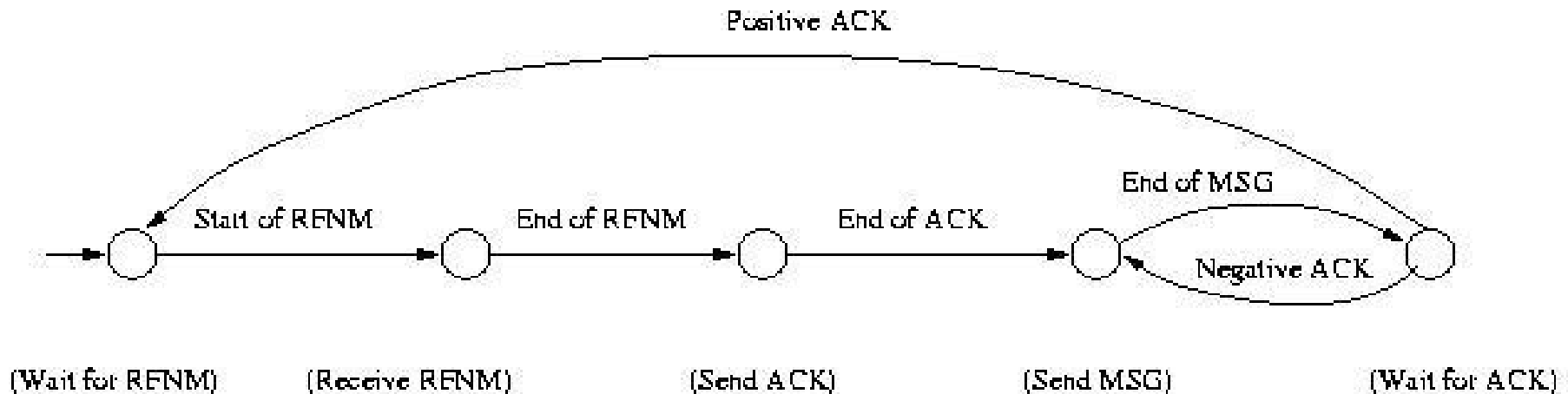
# Control Dependence

Traditional definition  
paths to *END*



# Control Dependence

Reactive definition  
oops ... there is no END



# Finally the Future

now, what my friends working on

# Future Ideas

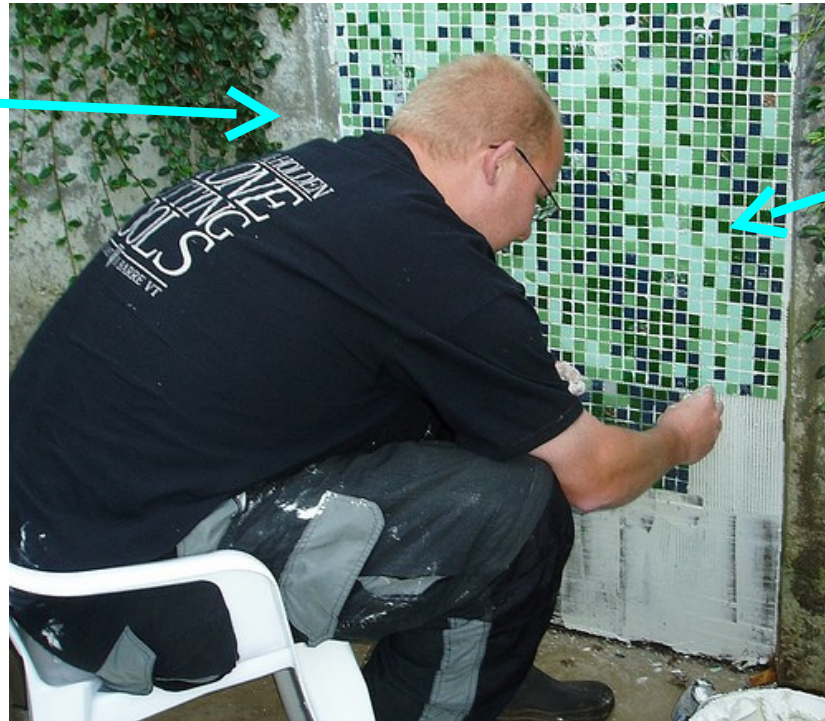
- Slicing design level concepts in code
- Slicing architecture descriptions
- Slicing UML
- Slicing comments and documentation
- Slicing from requirements to code and back
- **Slices as building blocks**

# Future Ideas

## *Program Composition*

Think of a program as composed of particular **syntactic entities**

Code Monkey



Program

# Future Ideas

## *Program Composition*

Dominant tiles 1960's  
***assembler instructions***

|     |       |      |
|-----|-------|------|
| ADD | CSECT |      |
|     | BALR  | 12,0 |
|     | USING | *,12 |
|     | AR    | 2,3  |
|     | BR    | 14   |

# Future Ideas

## *Program Composition*

Dominant tiles 1970's  
***statements***

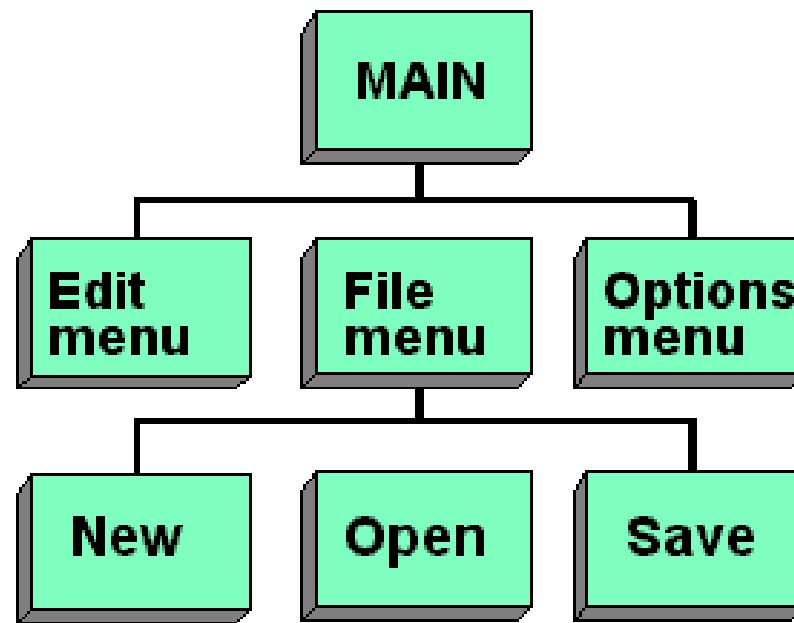
|    |               |
|----|---------------|
|    | if x 10,20,20 |
| 10 | x = x * -1    |
| 20 | continue      |

# Future Ideas

## *Program Composition*

Dominant tiles 1980's  
***functions***

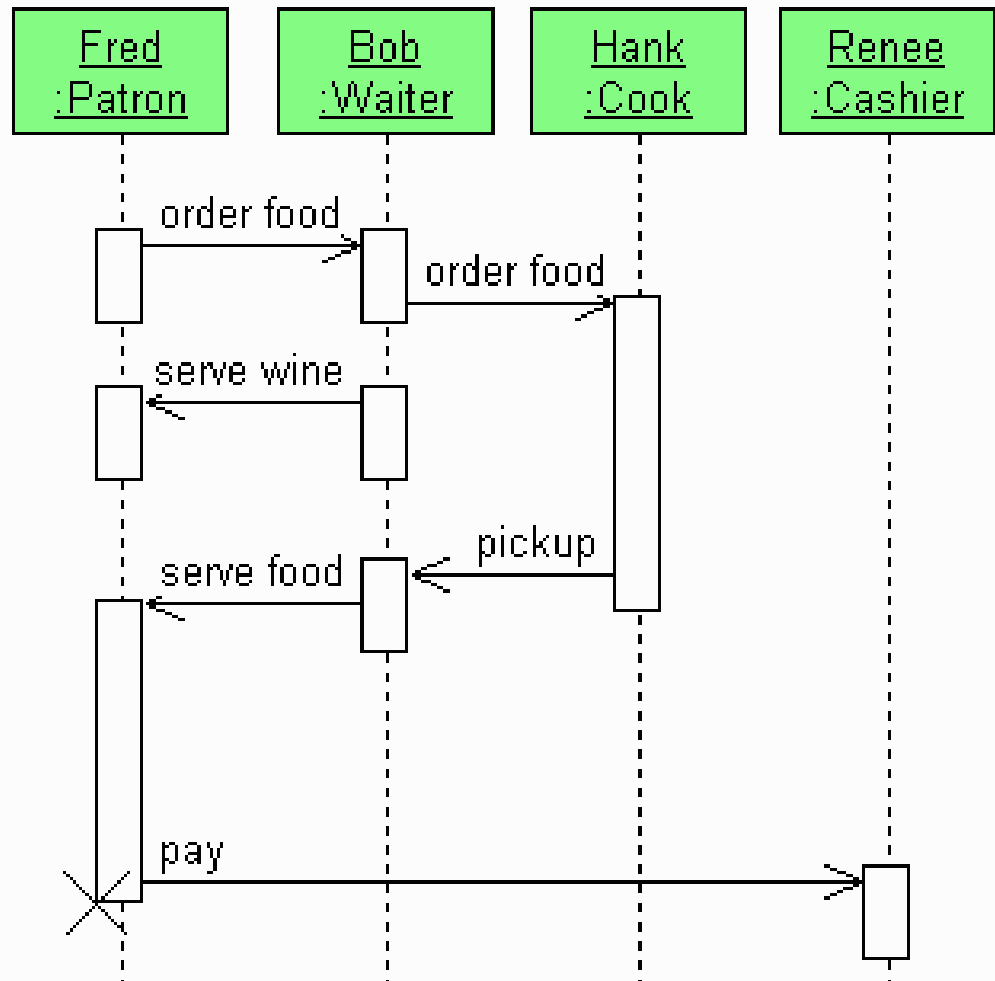
From Computer Desktop Encyclopedia  
© 1998 The Computer Language Co. Inc.



# Future Ideas

## *Program Composition*

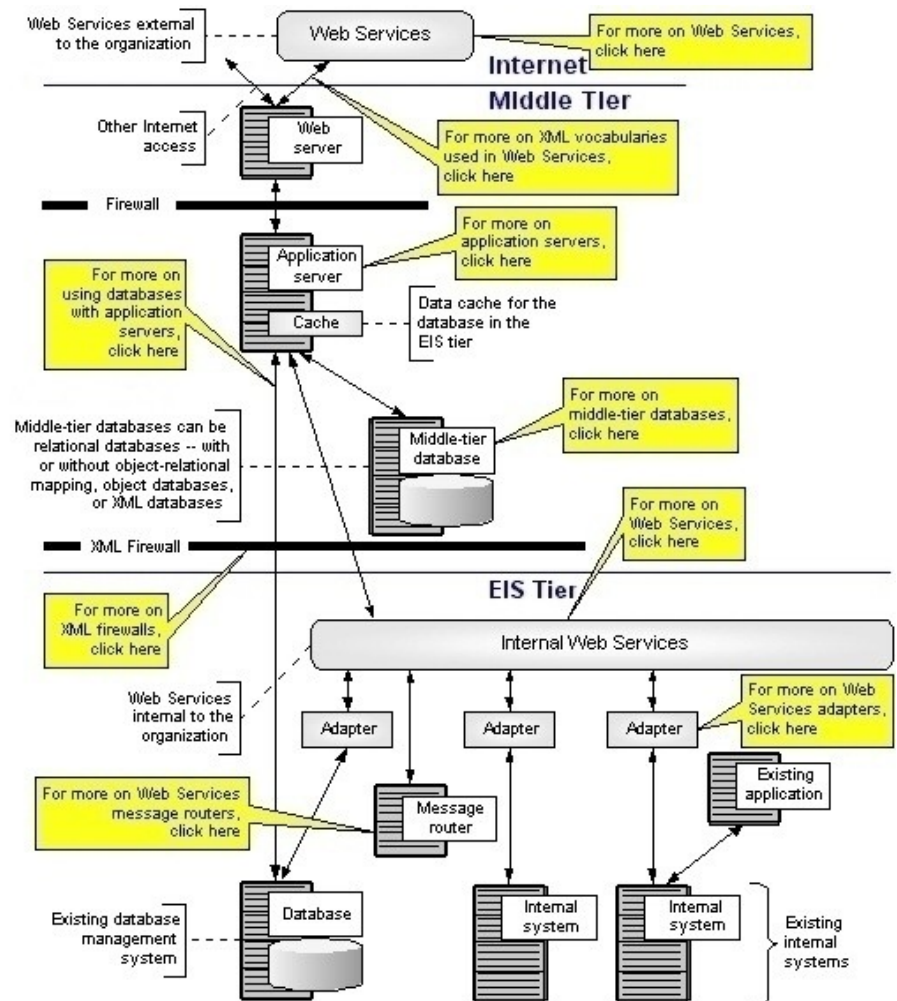
Dominant  
tiles 1990's  
*objects*



# Future Ideas

## *Program Composition*

“Dominant”  
tiles 2000’s  
*services*



# Future Ideas

## *Program Composition*

Dominant tiles of the future

***slices!***



# Questions?

## Sample Tool URLs

- [www.dsic.upv.es/users/elp/german/slicing](http://www.dsic.upv.es/users/elp/german/slicing)
- [kathrin.dagstuhl.de/05451/Materials2](http://kathrin.dagstuhl.de/05451/Materials2)
- [bandera.projects.cis.ksu.edu/papers/slicing.shtml](http://bandera.projects.cis.ksu.edu/papers/slicing.shtml)
- [www.ssw.unilinz.ac.at/Research/Projects/ProgramSlicing](http://www.ssw.unilinz.ac.at/Research/Projects/ProgramSlicing)
- [jslice.sourceforge.net](http://jslice.sourceforge.net)
- [hissa.ncsl.nist.gov/~jimmy/refs.html](http://hissa.ncsl.nist.gov/~jimmy/refs.html)
- [www.dagstuhl.de/en/program/calendar/semhlp/?semnr=05451](http://www.dagstuhl.de/en/program/calendar/semhlp/?semnr=05451)
- [www.gramatech.com](http://www.gramatech.com)

(largely from the first 200 hits of the web search '*program slicing tools*')