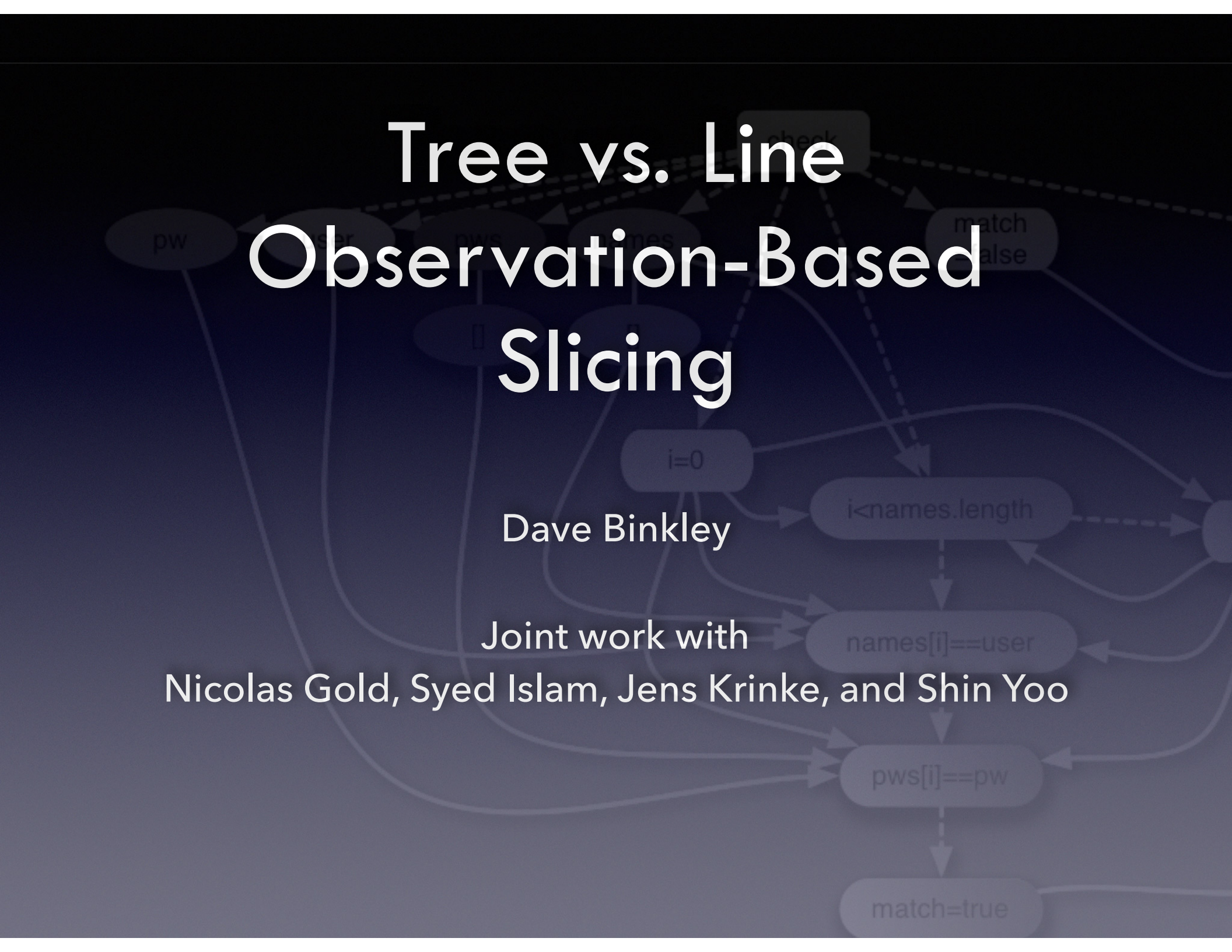
Tree vs. Line Observation-Based Slicing

Dave Binkley

Joint work with
Nicolas Gold, Syed Islam, Jens Krinke, and Shin Yoo



A Slice - Two Requirements

- Program P

```
prod = 1
sum = 0
i = 1
while (i < 10)
    prod *= i
    sum += i
    i += 1
endwhile
print sum
print prod
```

A Slice - Two Requirements

- Program P

```
prod = 1
sum = 0
i = 1
while (i < 10)
    prod *= i
    sum += i
    i += 1
endwhile
print sum
print prod
```

- A slice of P
- Syntactic subset

```
sum = 0
i = 1
while (i < 10)

    sum += i
    i += 1
endwhile
print sum
```

A Slice - Two Requirements

- Program P

```
prod = 1
sum = 0
i = 1
while (i < 10)
    prod *= i
    sum += i
    i += 1
endwhile
print sum
print prod
```

- A slice of P
- Syntactic subset

```
sum = 0
i = 1
while (i < 10)

    sum += i
    i += 1
endwhile
print sum
```

- Same Semantics
- Sequence of values

<45>

<362880>

Static Slicing (Weiser)

A **static** slice S of program P on slicing criterion C is any executable program where

1. S can be obtained from P
   by **deleting** zero or more statements from P.
2. Whenever P halts on **any input i**
   with state trajectory T,
   then S also halts on input i with state trajectory T'
   and $\text{PROJ}_C(T)$ is the **same** as $\text{PROJ}_C(T')$,

where PROJ_C projects values associated with C.

Dynamic Slicing

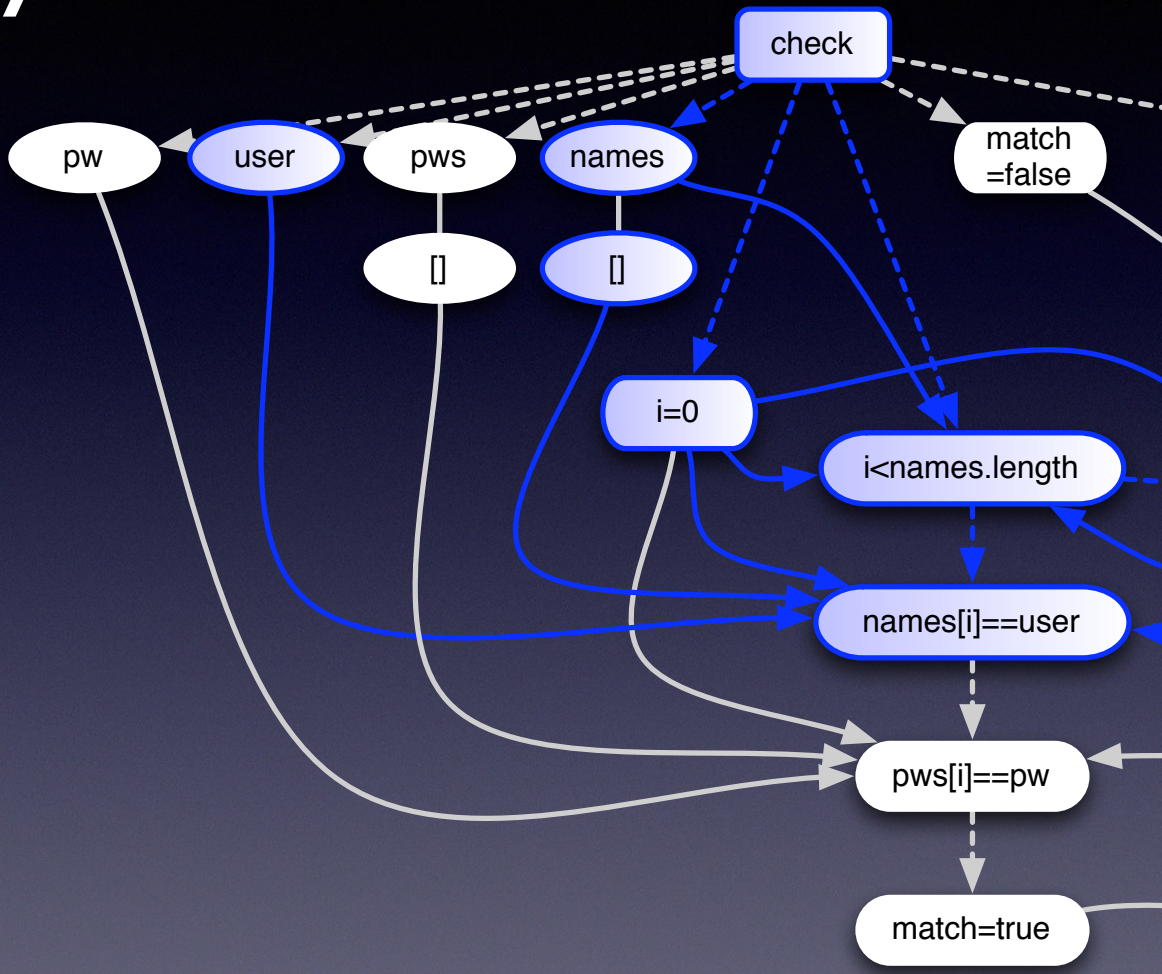
A **dynamic** slice *S* of program *P* on slicing criterion C is any executable program where

1. *S* can be obtained from *P*
   by **deleting** zero or more statements from *P*.
2. Whenever *P* halts on **input i ∈ I**
   with state trajectory *T*,
   then *S* also halts on input *i* with state trajectory *T′*
   and $\text{PROJ}_C(T)$ is the **same** as $\text{PROJ}_C(T')$,

where PROJ_C projects values associated with C.

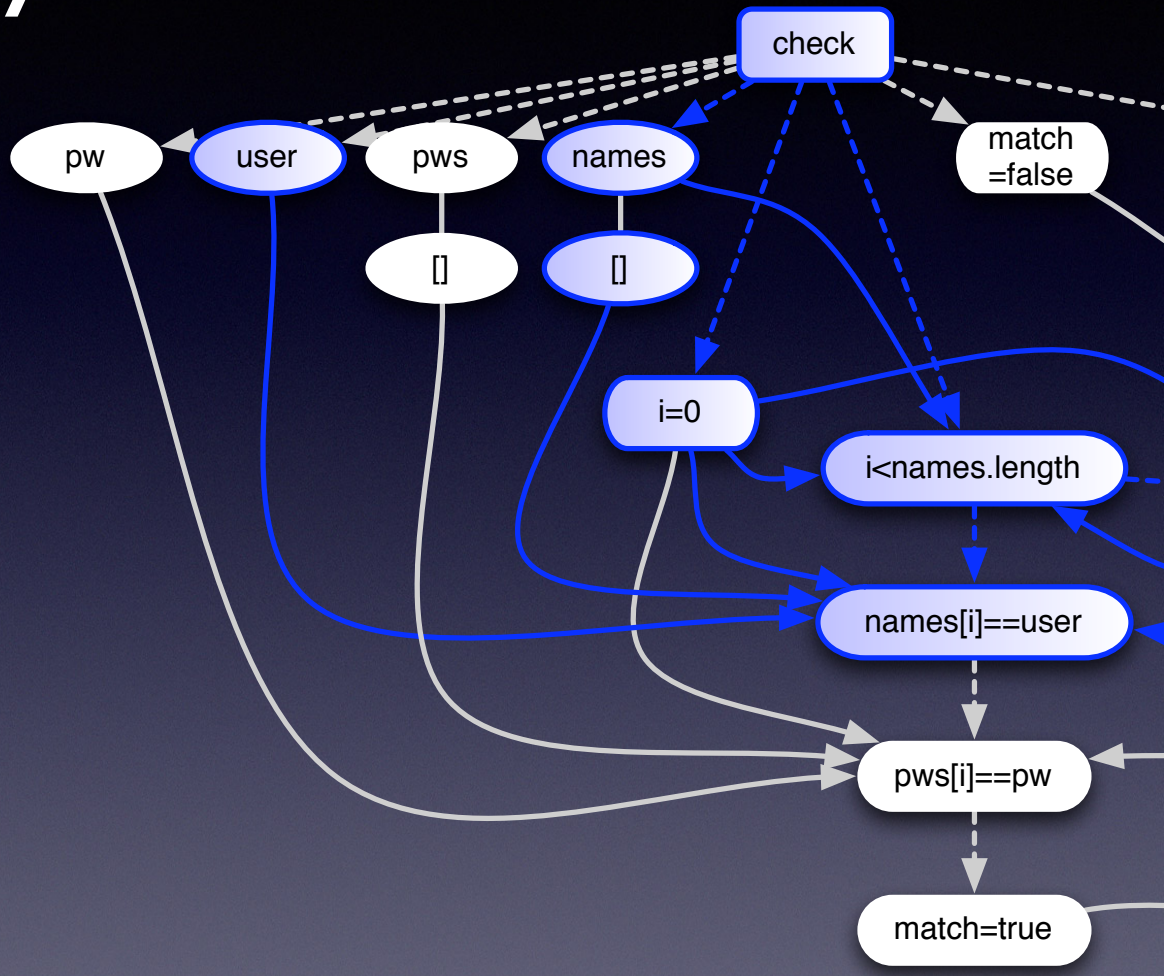
Slicing is easy!

Slicing is easy!



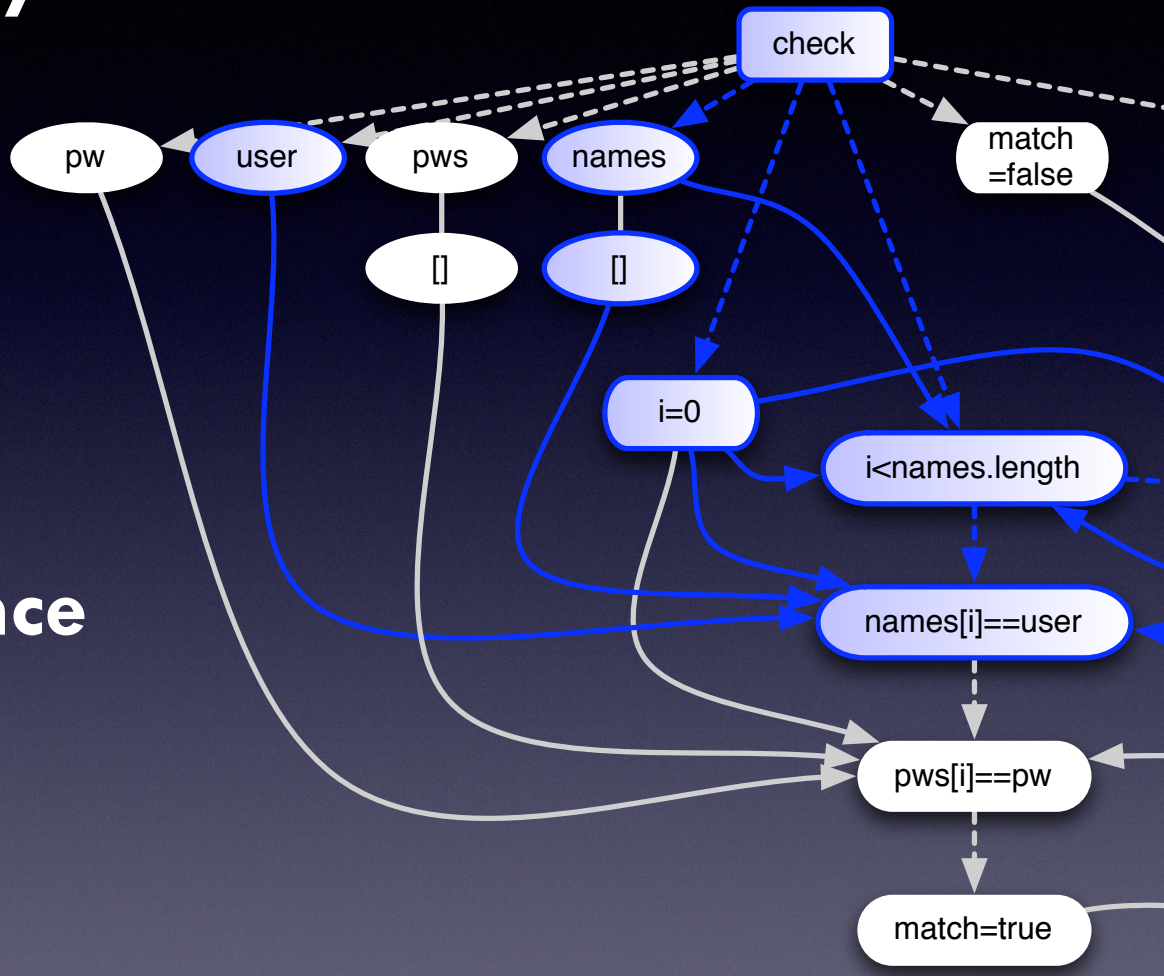
Slicing is easy!

- … just traverse the dependences (backward)



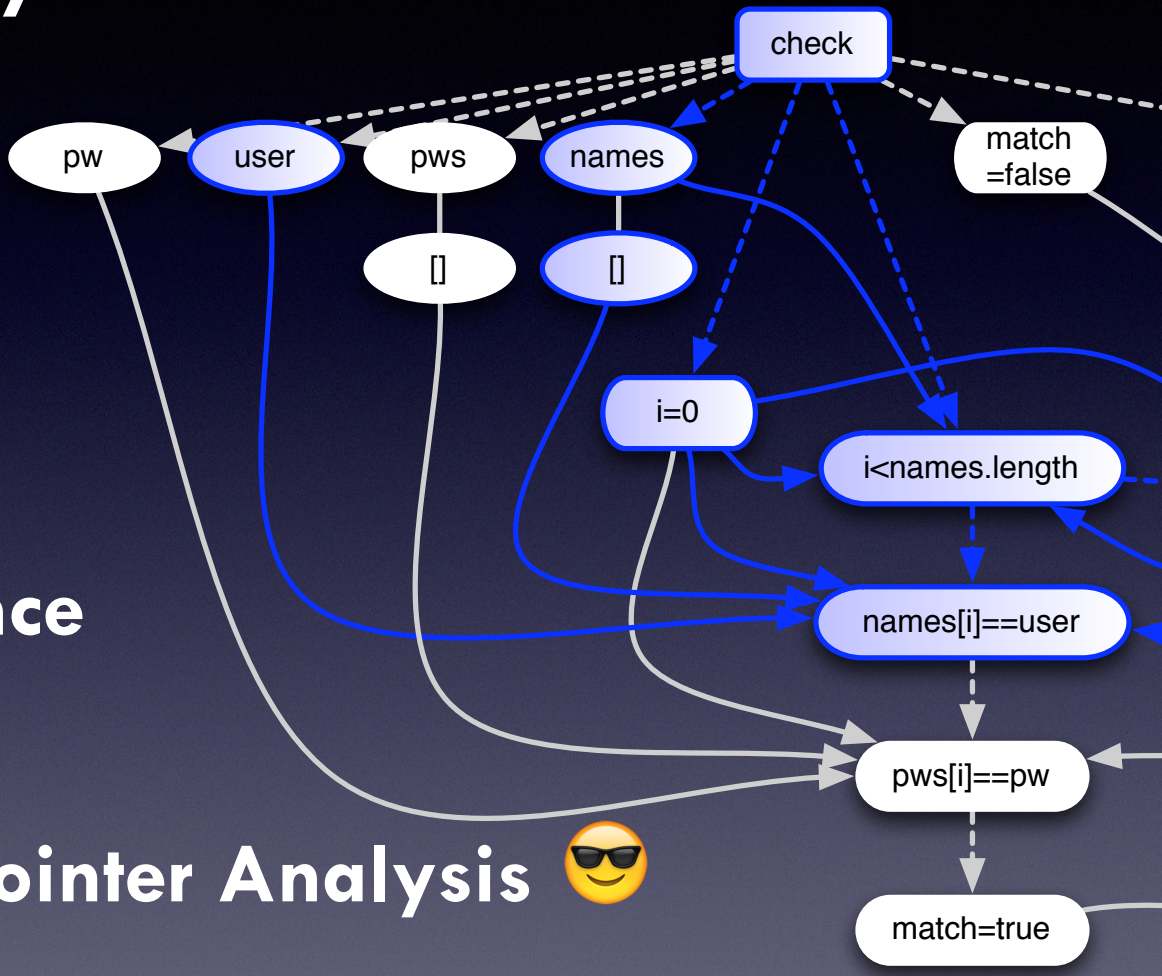
Slicing is easy!

- … just traverse the dependences (backward)
- Oh wait, **dependence analysis** is hard!



Slicing is easy!

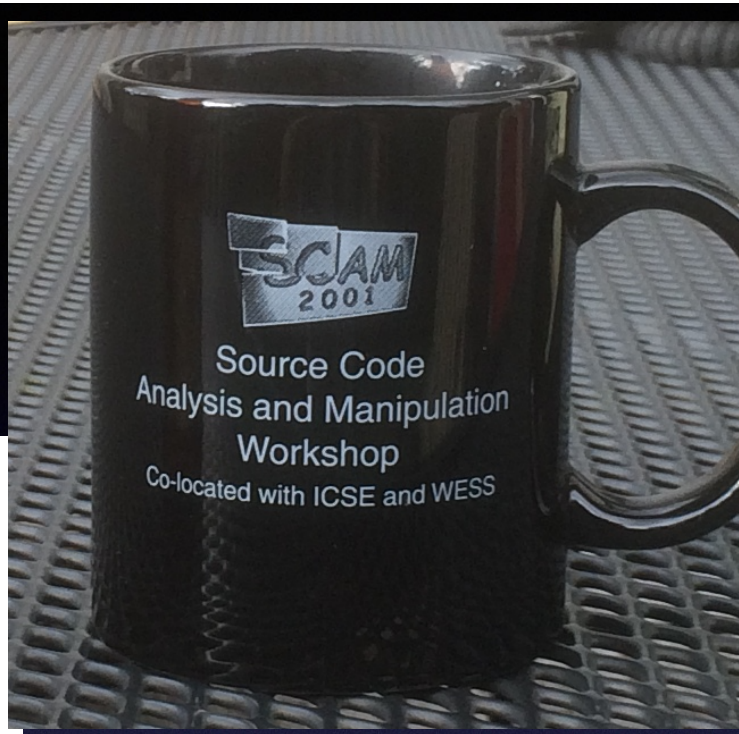
- … just traverse the dependences (backward)
- Oh wait, **dependence analysis** is hard!
- Ever had a go at **Pointer Analysis** 😎

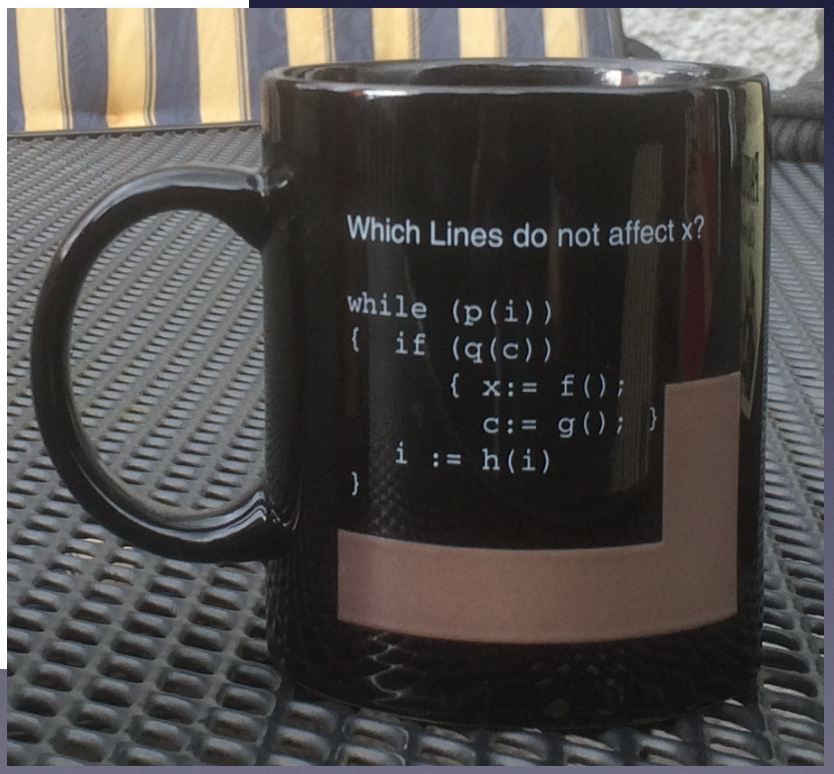


SCAM 2001

Which Lines do not affect x?

```
 1 int mug(int i, int c, int x)
 2 {
 3   while (p(i))
 4   {
 5     if (q(c))
 6     {
 7       x = f();
 8       c = g();
 9     }
10     i = h(i);
11   }
12   printf("@%d\n", x);
13 }
```

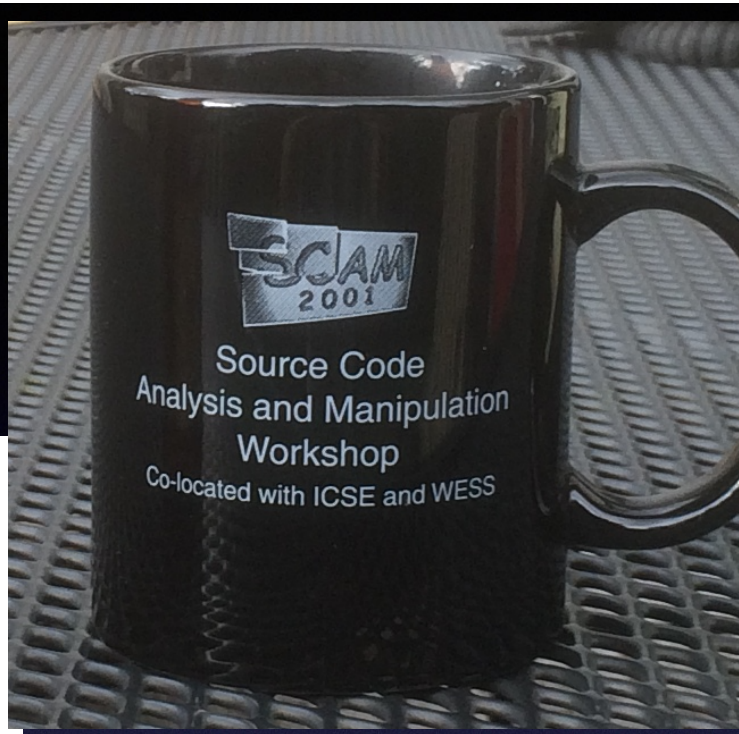


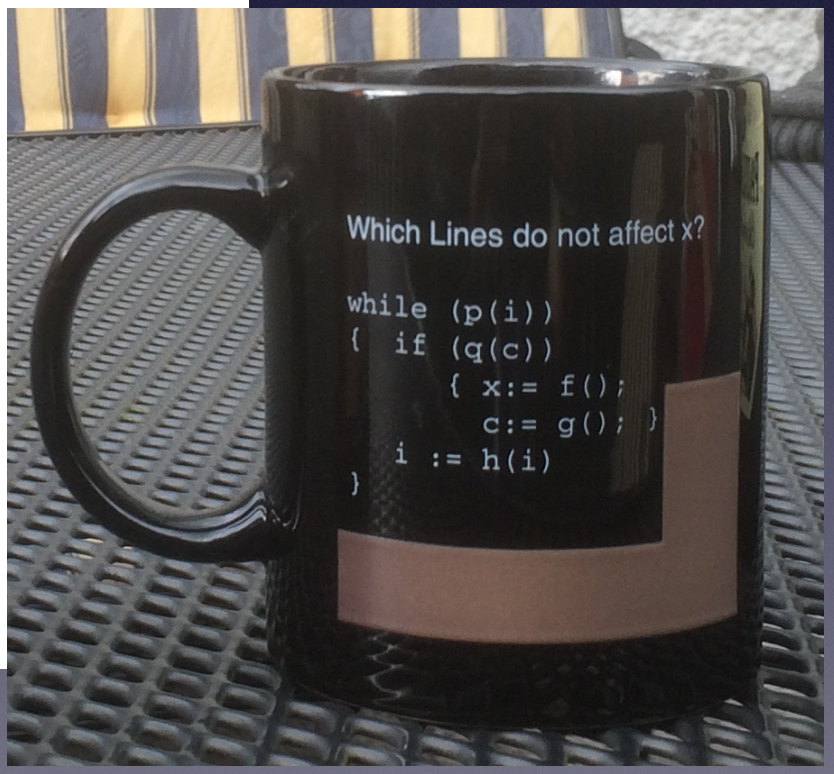


SCAM 2001

Which Lines do not affect x?

```
 1 int mug(int i, int c, int x)
 2 {
 3   while (p(i))
 4   {
 5     if (q(c))
 6     {
 7       x = f();
 8       c = g();
 9     }
10     i = h(i);
11   }
12   printf("@%d\n", x);
13 }
```

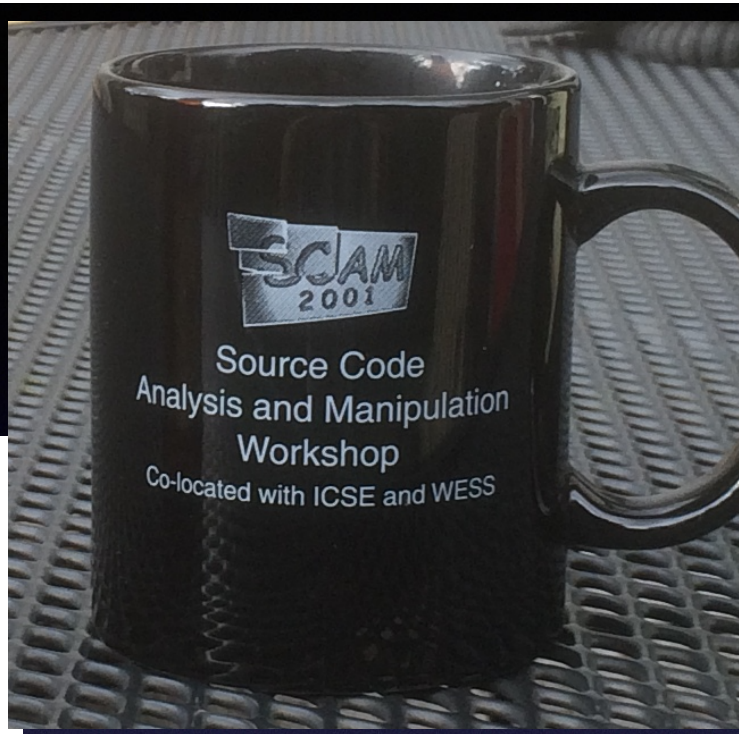


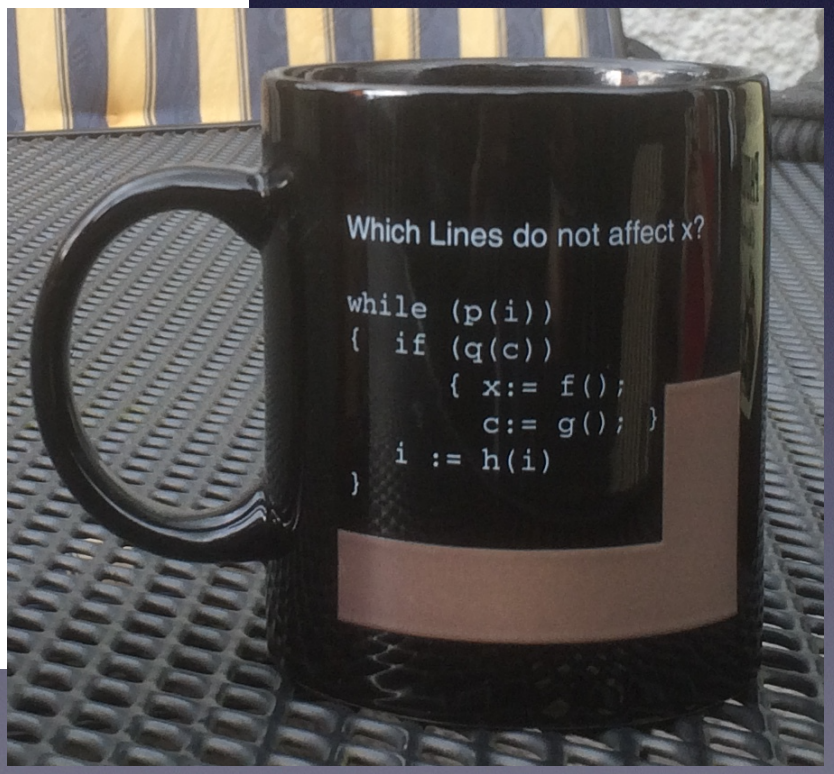


SCAM 2001

Which Lines do not affect x?

```
 1 int mug(int i, int c, int x)
 2 {
 3   while (p(i))
 4   {
 5     if (q(c))
 6     {
 7       x = f();
 8       c = g();
 9     }
10     i = h(i);
11   }
12   printf("@%d\n", x);
13 }
```

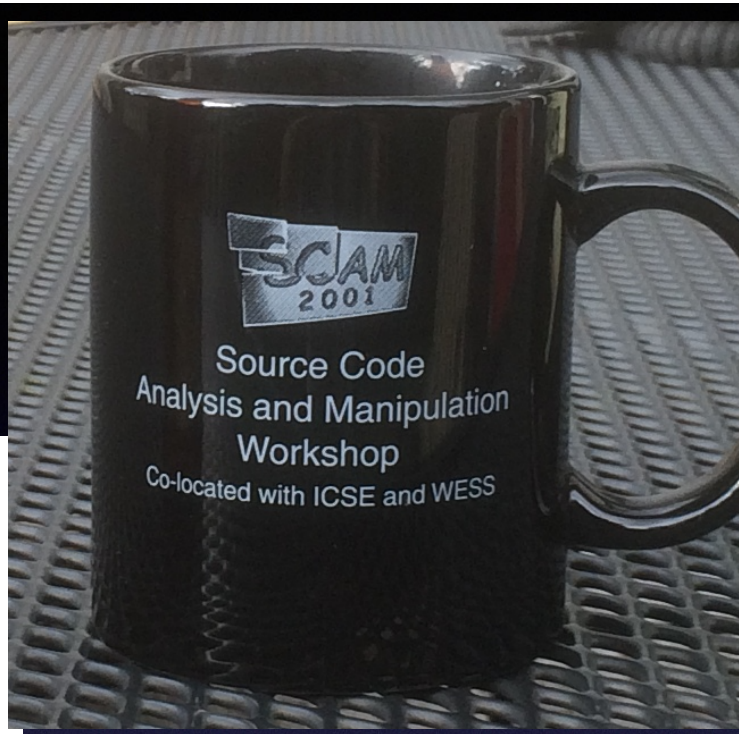


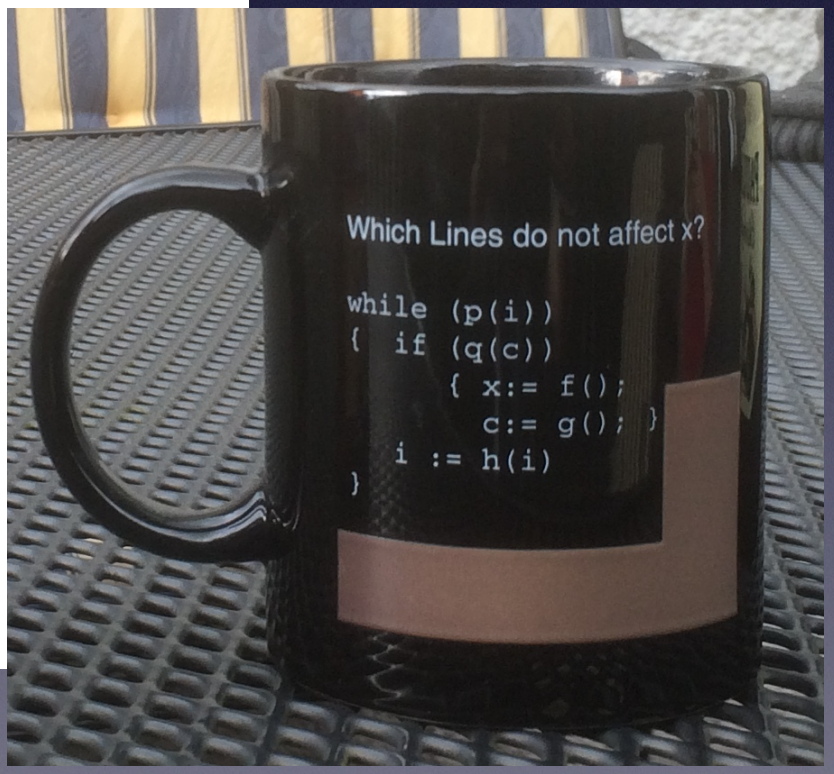


SCAM 2001

Which Lines do not affect x?

```
 1 int mug(int i, int c, int x)
 2 {
 3   while (p(i))
 4   {
 5     if (q(c))
 6     {
 7       x = f();
 8       c = g();
 9     }
10     i = h(i);
11   }
12   printf("@%d\n", x);
13 }
```





Observation-Based Slicing

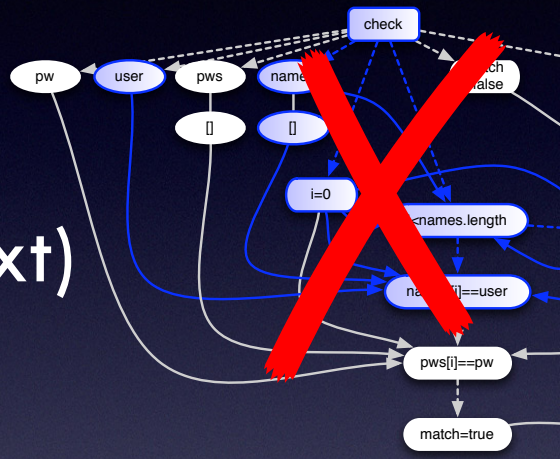
- side-steps dependence analysis
- is language independent
- slices multiple languages
- supports binary components and libraries
- produces executable slices

Observation-Based Slicing

- side-steps dependence analysis
- is language independent
- slices multiple languages
- supports binary components and libraries
- produces executable slices

All without any dependence analysis!

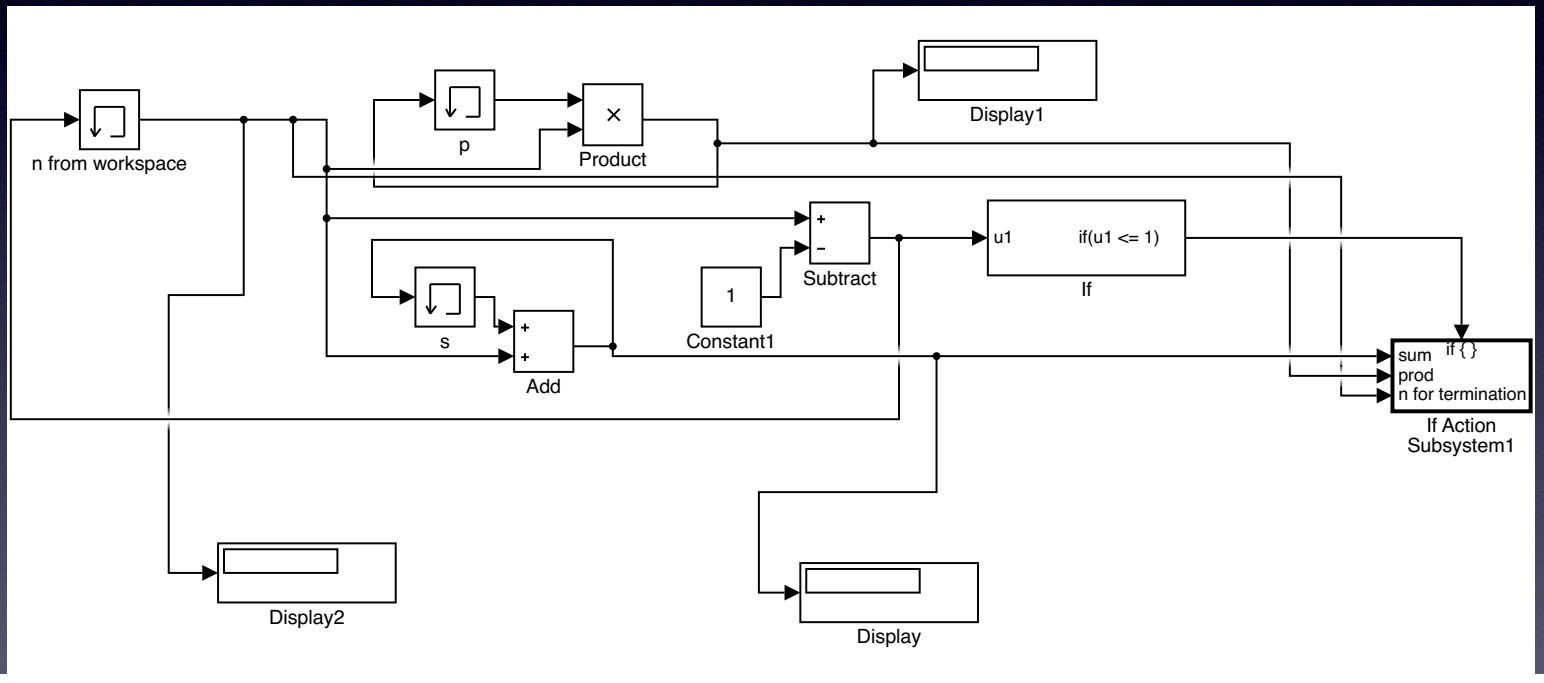
Here's how it works



- delete something (e.g., a line of text)
- execute candidate slice
- observe the behaviour of criterion
- accept deletion if behaviour is unchanged
- repeat until no more deletions are possible

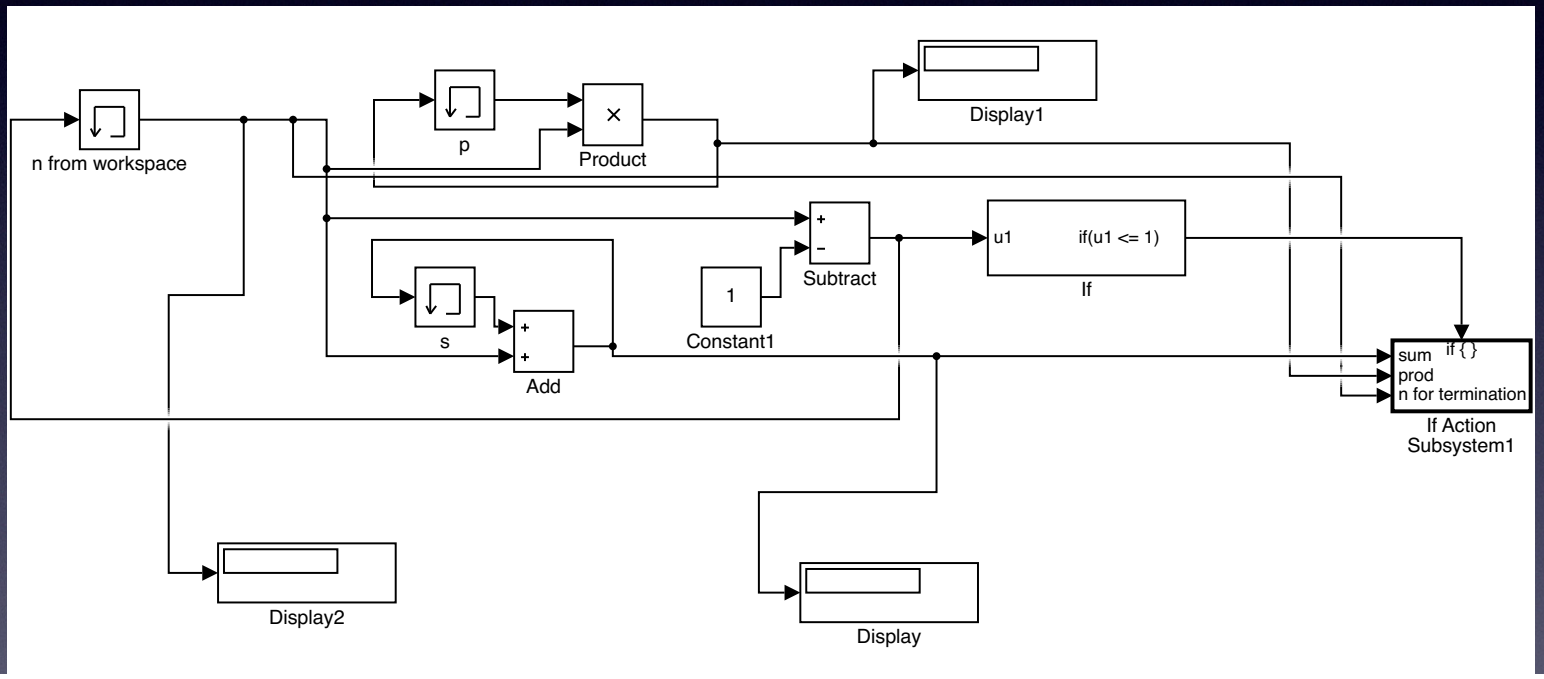
Yeabut

Here's my source code



Yeabut

Here's my source code



😀 ORBS can slice *Simulink* models

😕 but is not natural

- 😀 ORBS can slice *Simulink* models
- 😕 but is not natural

```
<Block BlockType="SubSystem"
        Name="while loop body" SID="104">
  <P Name="Ports">[2, 2, 0, 0, 0, 0, 0, 1]</P>
  <P Name="Position">[395, 222, 465, 258]</P>
  <P Name="RequestExecContextInheritance">off</P>
   ....
```

Enter Tree-ORBS

TORBS can slice Simulimk's XML trees!

Enter Tree-ORBS

TORBS (

Algorithm 2: Core of the Tree-ORBS Slicer

T-ORBS_CORE$(T, O, \mathcal{I})$

Input: Current Tree T, the criterion consisting of observer O, and input set $\mathcal{I}$

Output: Updated Tree, T

(1) $q \leftarrow$ APPEND$(empty_queue, start_node(T))$
(2) **while** $\neg$ EMPTY(q)
(3) $c \leftarrow$ DEQUEUE(q)
(4) $T' \leftarrow$ DELETE(T, c)
(5) $V' \leftarrow O(T', \mathcal{I})$
(6) **if** $V = V'$
(7) $T \leftarrow T'$
(8) **else**
(9) $q \leftarrow$ APPEND$(q,$ CHILDREN$(c))$
(10) **return** T

Enter Tree-ORBS

TORBS can slice Simulimk's XML trees!

```
if (a > b)
  m = a;
else
  m = b;
```

Here's a tree

```
<if>if
 <condition>(...)</condition>
 <then> <expr_stmt> ... </expr_stmt></then>
 <else>else <expr_stmt> ...</expr_stmt></else>
</if>
```

Initial Study

Program	LoC	SLoC	Slices
Known Semantics			
sumprod	20	16	8
wc	128	70	17
mug	73	62	16
mbe	82	62	12
Exhaustively Sliced			
tcas	185	141	43
schedule2	368	291	78
schedule	465	313	58
totinfo	573	347	54
printtokens2	638	407	75
replace	658	541	309
printtokens	895	569	81
Production Systems			
ed	3 062	2 393	1
byacc	7 760	6 615	1
bash	68 230	48 339	1
Non-C Systems			
Hanoi.java	171	158	1
permutation.java	658	3091	1
concordance.c++	1490	1033	1

So, slice the text or the tree?

RQ1: How do ORBS and T-ORBS slices compare quantitatively?

RQ2: How do the slices produced by ORBS and T-ORBS compare qualitatively?

RQ3: What impact does implementation have on the time taken to compute a slice?

Sub-Line (stmt) Dependence 😀

```
typedef enum Boolean { FALSE = 0, TRUE = 1, FAIL = 0,
SUCCEED = 1, OK = 1, NO = 0, YES = 1, NOMSG = 0,
MSG = 1, OFF = 0, ON = 1 } BOOLEAN;}
```

Sub-Line (stmt) Dependence 😀

ORBS retains this entire line

```
typedef enum Boolean { FALSE = 0, TRUE = 1, FAIL = 0,
SUCCEED = 1, OK = 1, NO = 0, YES = 1, NOMSG = 0,
MSG = 1, OFF = 0, ON = 1 } BOOLEAN;}
```

Sub-Line (stmt) Dependence 😀

ORBS retains this entire line

```
typedef enum Boolean { FALSE = 0, TRUE = 1, FAIL = 0,
SUCCEED = 1, OK = 1, NO = 0, YES = 1, NOMSG = 0,
MSG = 1, OFF = 0, ON = 1 } BOOLEAN;}
```

TORBS exploits sub-statement dependence

```
typedef enum { OK = 1, NO = 0, YES  } BOOLEAN;
```

TORBS Preserves Structure 😞

```
#define OLEV      600        /* in feets/minute */
...
  enabled = High_Confidence && (OLEV < ...
  if (enabled && ...
  {
      need_downward_RA = ...   // in slice
```

ORBS Slice

TORBS Preserves Structure 😀

```
for(i=1; i<=10; i++)
 {
  sum = sum + i;
  prod = prod * i;
 }
printf("at end i = %d\n", i);
printf("\norbs:%d\n", i);  //slice here w.r.t. i
```

TORBS Preserves Structure 😀

```
for(i=1; i<=10; i++)
 {
 }

printf("\norbs:%d\n", i);  //slice here w.r.t. i
```

ORBS Does Not 😞

```
for(i=1; i<=10; i++)
 {
  sum = sum + i;
  prod = prod * i;
 }
printf("at end i = %d\n", i);
printf("\norbs:%d\n", i);  //slice here w.r.t. i
```

ORBS Does Not 😞

```
for(i=1; i<=10; i++)
    printf("at end i = %d\n", i);
printf("\norbs:%d\n", i);  //slice here w.r.t. i
```

In Comparison

```
for(i=1; i<=10; i++)
   printf("at end i = %d\n", i);
printf("\norbs:%d\n", i);  //slice here w.r.t. i
```

```
   for(i=1; i<=10; i++)
   {
   }
   printf("\norbs:%d\n", i);  //slice here w.r.t. i
```

(A similar capture example)

```
while (scanf("%c", &c) == 1)
{
  if (isletter(c))
  {
...

int isletter(char c)
{
  printf("\norbs:%c\n",c); //slice here w.r.t. c
```

(A similar capture example)

```
while (scanf("%c", &c) == 1)
{
  if (isletter(c))
  {
...

int isletter(char c)
{
  printf("\norbs:%c\n",c); //slice here w.r.t. c
```

```
while (scanf("%c", &c) == 1)
  printf("\norbs:%c\n",c); //slice here w.r.t. c
```

ORBS can handle “separate” trees 😀

#ifdef DEBUG
#endif

ORBS Preserves Structure 😀

```
if (q(k))
{
  k = f1(k)
  printf("\norbs:%d\n", k)       // slice here
```

ORBS Preserves Structure 😀

ORBS Slice
(yea it's
the same)

```
if (q(k))
{
  k = f1(k)
  printf("\norbs:%d\n", k)       // slice here
```

ORBS Preserves Structure 😀

ORBS Slice
(yea it's
the same)

```
if (q(k))
{
  k = f1(k)
  printf("\norbs:%d\n", k)       // slice here
```

TORBS Slice

```
if (q(k))
{
  k = f1(   )
  printf("\norbs:%d\n", k)       // slice here
```

Bonus:
ORBS Preserves Structure😞

```
{
   inword = 1;
}
```

Bonus:
ORBS Preserves Structure😞

```
{
   inword = 1;
}
```

ORBS Slice
(yea it's
the same)

```
inword = 1;
```

TORBS Slice
(subtree replacement)

TORBS can Bog Down 😞

d = sqrt(b*b - 4*a*c)

TORBS can Bog Down 😞

```
d =  sqrt(b*b - 4*a*c)
```

ORBS tries to delete the line

```
d =  sqrt(_*b - 4*a*c)
d =  sqrt(b_b - 4*a*c)
d =  sqrt(b*_ - 4*a*c)
d =  sqrt(b*b _ 4*a*c)
d =  sqrt(b*b - _*a*c)
d =  sqrt(b*b - 4_a*c)
d =  sqrt(b*b - 4*a_c)
d =  sqrt(b*b - 4*a*_)
```

TORBS **also** tries to delete

Summary

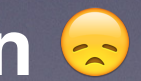
TORBS - sub-line (stmt) dependence 

TORBS preserves structure 

TORBS preserves structure 

ORBS can handle “separate” trees 

ORBS preserves structure 

TORBS can Bog Down 

TORBS preserves structure 😞

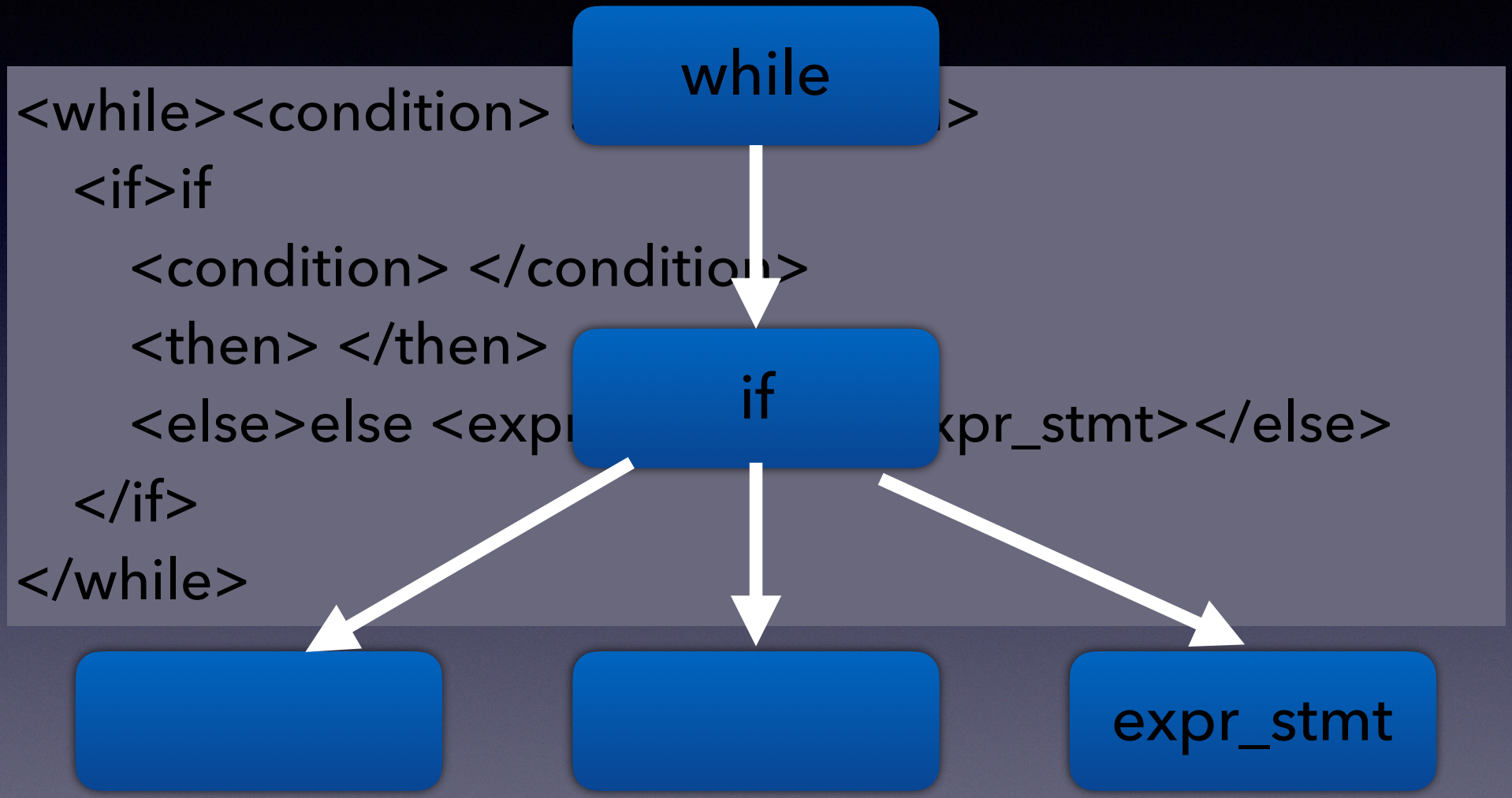
```
while (p(j))
{
    if ((k))
    {
    }
    else
    {
        j = f3();
        printf("\norbs:%d\n", j);
    }
}
```

Subtree Replacement

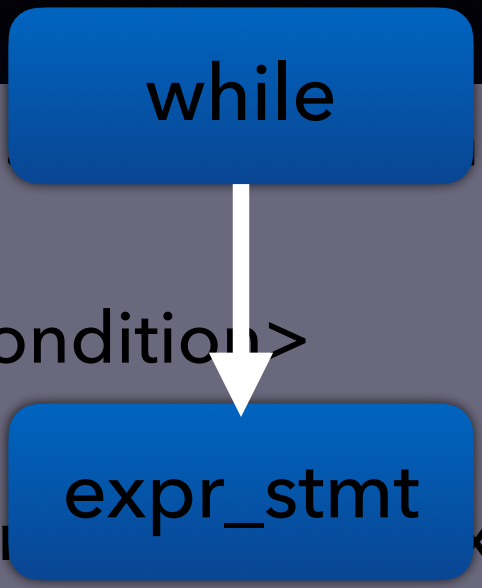
Subtree Replacement

```
<while><condition> ... </condition>
  <if>if
    <condition> </condition>
    <then> </then>
    <else>else <expr_stmt> ...</expr_stmt></else>
  </if>
</while>
```

Subtree Replacement

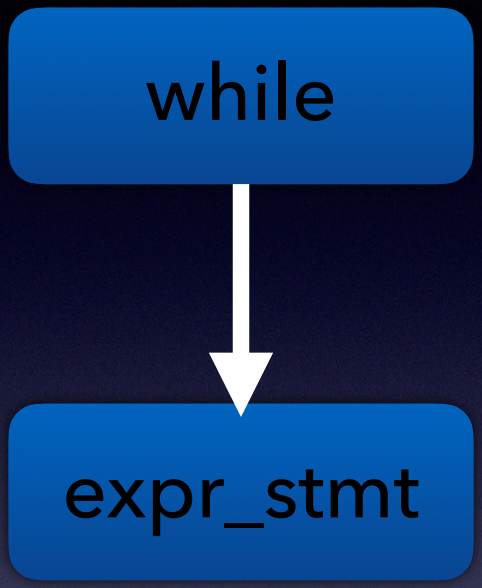


Subtree Replacement




```
<while><condition>
  <if>if
    <condition> </condition>
    <then> </then>
    <else>else <expr_stmt></else>
  </if>
</while>
```

Subtree Replacement

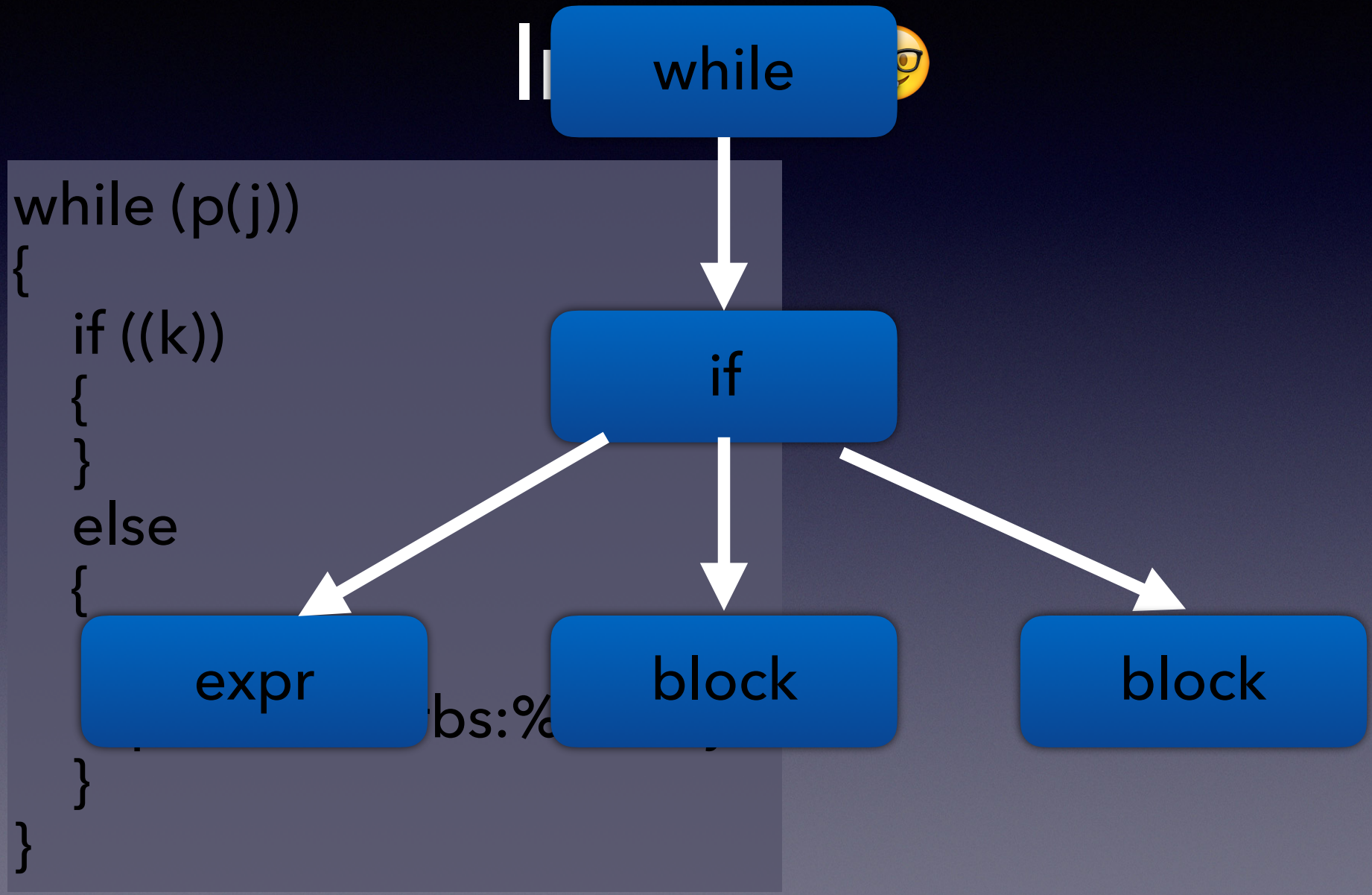



```
<while><condition> ... </condition>
      <expr_stmt> ...</expr_stmt></else>
</while>
```

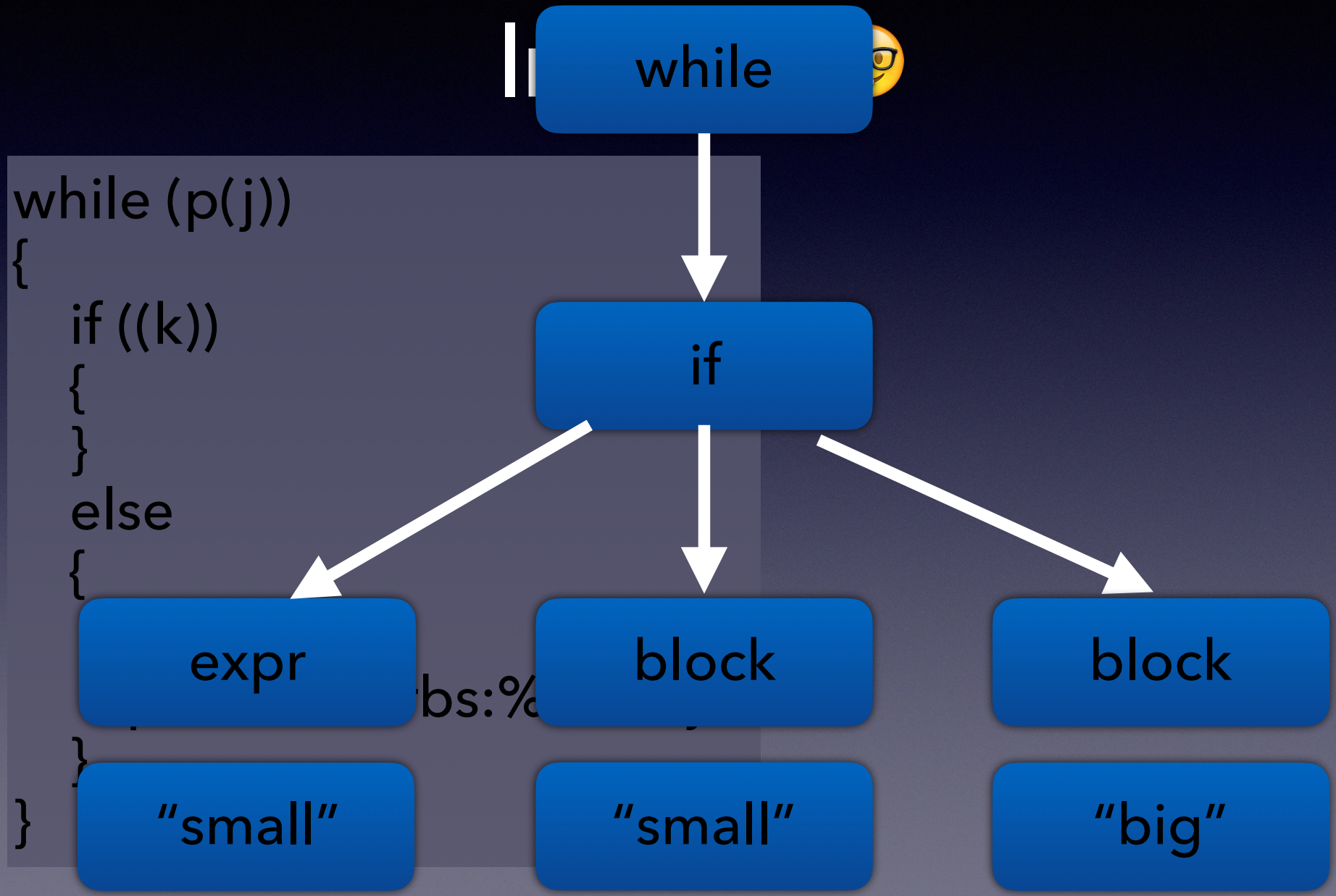
Subtree Replacement
In fact 🤓

```
while (p(j))
{
    if ((k))
    {
    }
    else
    {
        j = f3();
        printf("\norbs:%d\n", j);
    }
}
```

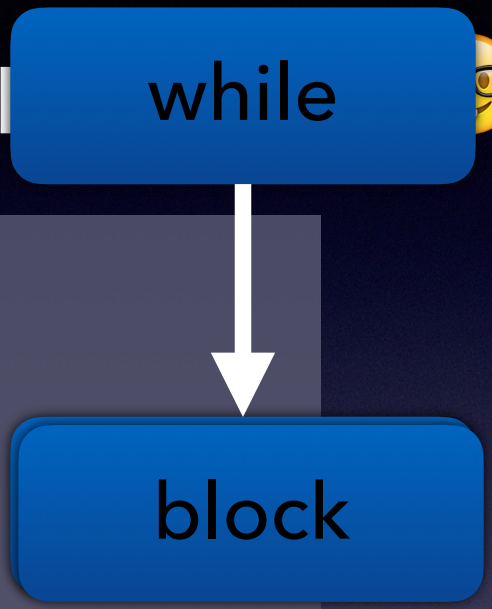
Subtree Replacement



Subtree Replacement

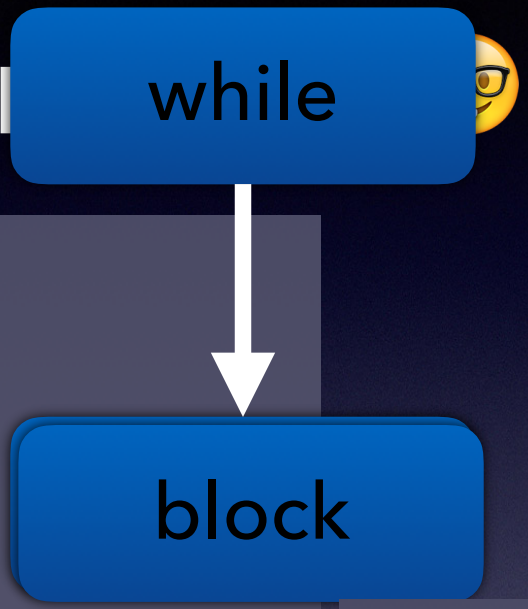


Subtree Replacement




```
while (p(j))
{
    if ((k))
    {
    }
    else
    {
        j = f3();
        printf("\norbs:%d\n", j);
    }
}
```

Subtree Replacement




```
while (p(j))
{
    if ((k))
    {
    }
    else
    {
        j = f3();
        printf("\norbs:%d\n", j);
    }
}
```

```
while (p(j))
{
    j = f3();
    printf("\norbs:%d\n", j);
}
```

TORBS can Bog Down 😞

Subtree
Minimum
Size

TORBS can Bog Down 😞

```
d =  sqrt(_*b - 4*a*c)
d =  sqrt(b_b - 4*a*c)
d =  sqrt(b*_ - 4*a*c)
d =  sqrt(b*b _ 4*a*c)
d =  sqrt(b*b - _*a*c)
d =  sqrt(b*b - 4_a*c)
d =  sqrt(b*b - 4*a_c)
d =  sqrt(b*b - 4*a*_)
```

Subtree Minimum Size

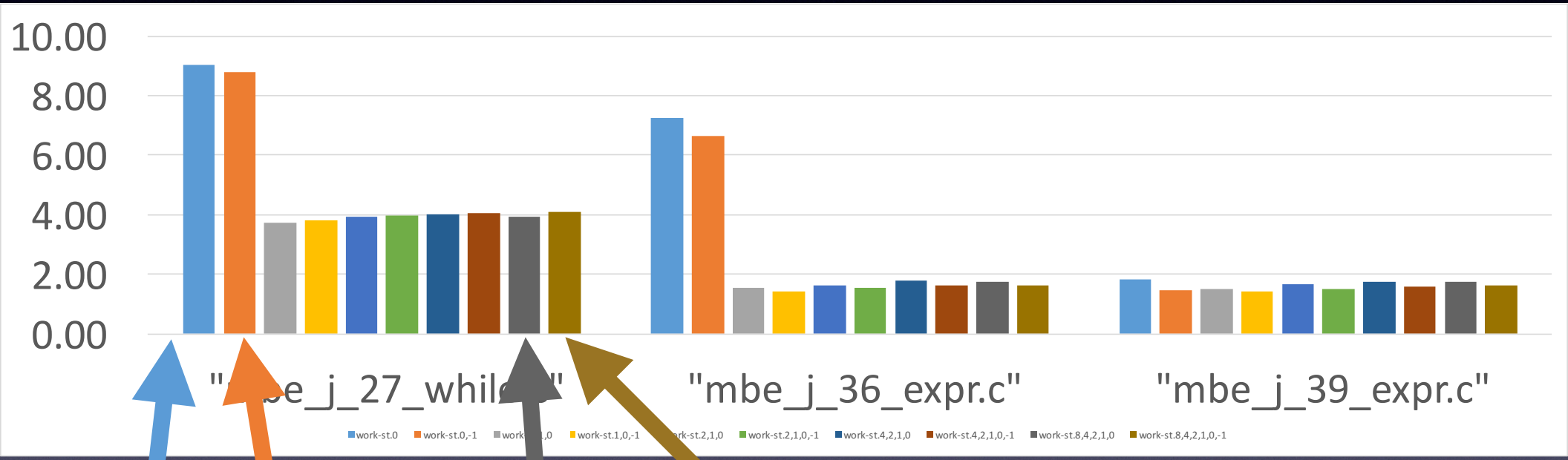
“-st”

-st 0	unchanged from previous
-st 0,-1	post “slice” run subtree replacement
-st 1,0	perform no sublime deletions on first pass

Timing Impact Study

-st 0
-st 0,-1
-st 1,0
-st 1,0,-1
-st 2,1,0
-st 2,1,0,-1
-st 4,2,1,0
-st 4,2,1,0,-1
-st 8,4,2,1,0
-st 8,4,2,1,0,-1

CPU Time



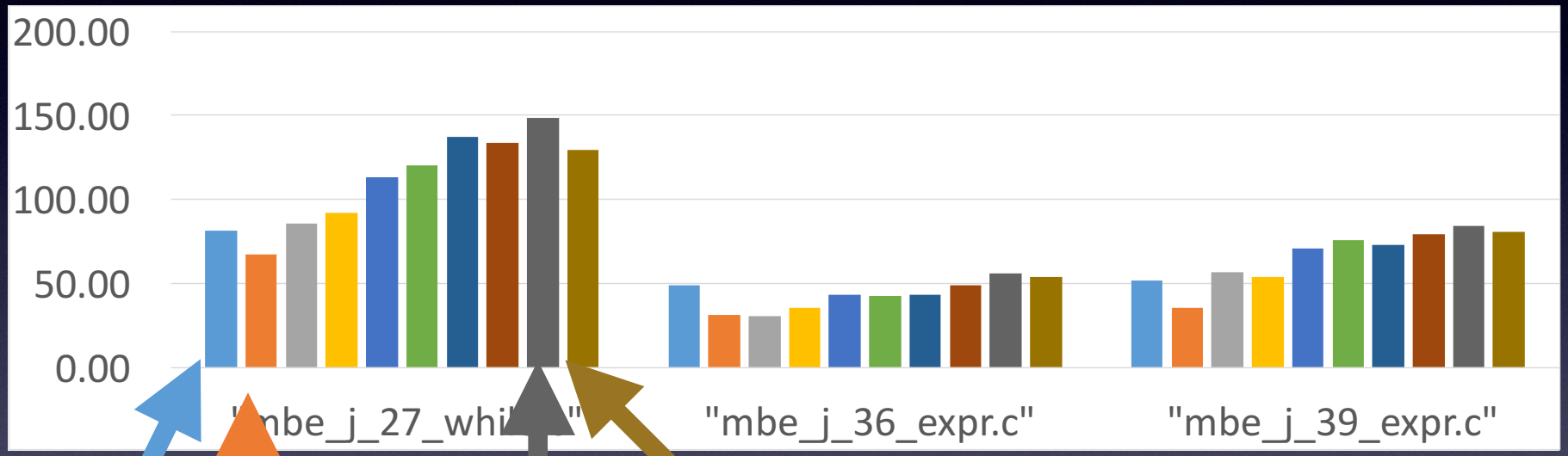
-st 0

-st 0,-1

-st 8,4,2,1,0

-st 8,4,2,1,0,-1

Wall Clock Time



-st 0

-st 0,-1

-st 8,4,2,1,0

-st 8,4,2,1,0,-1

Current Challenge — GMAT

Language	files	blank	comment	code
C++	3011	271564	363278	1208448
HTML	1912	76587	3850	595289
C	3005	360893	701089	447398
C/C++ Header	3734	132486	309162	322258
Bourne Shell	193	24265	20054	136832
Fortran 77	8	1222	6382	68518
m4	65	6538	2006	55250
Objective C++	133	7366	5580	30667
XML	207	3101	1433	25093
make	74	956	1194	4440
Python	25	967	712	3732
XSD	79	448	97	2640
CMake	32	437	977	2385
XSLT	15	291	124	2078
CSS	6	287	47	1458
awk	3	93	393	783
DOS Batch	21	274	271	780
Teamcenter def	5	5	0	758
Perl	6	205	242	753
C Shell	6	294	665	662
MATLAB	33	175	964	662
Assembly	2	45	84	485
DTD	12	89	39	455
Bourne Again Shell	4	58	71	221
Lua	1	9	22	204
Javascript	4	17	34	144
Verilog-SystemVerilog	1	7	0	62
MXML	1	6	10	43
YAML	1	6	4	28
SUM:	12599	888691	1418784	2912526

Current Challenge — GMAT

Language	files	blank	comment	code
C++	3011	271564	363278	1208448
HTML	1912	76587	3850	595289
C	3005	360893	701089	447398
C/C++ Header	3734	132486	309162	322258
Bourne Shell	193	24265	20054	136832
Fortran 77	8	1222	6382	68518
m4	65	6538	2006	55250
Objective C++	133	7366	5580	30667
XML	207	3101	1433	25093
make	74	956	1194	4440
Python	25	967	712	3732
XSD	79	448	97	2640
CMake	32	437	977	2385
XSLT	15	291	124	2078
CSS	6	287	47	1458
awk	3	93	393	783
DOS Batch	21	274	271	780
Teamcenter def	5	5	0	758
Perl	6	205	242	753
C Shell	6	294	665	662
MATLAB	33	175	964	662
Assembly	2	45	84	485
DTD	12	89	39	455
Bourne Again Shell	4	58	71	221

files	blank	comment	code
12599	888691	1418784	2912526

Wait What??

-st 1,0

```
#include <stdio.h>
  c = (int) strtol(argv[2]);
```

-st 0

```
#include <stdlib.h>
  c = (int) strtol(argv[2], NULL, 10);
```

Thanks