Feedback Directed Optimization

Shambo Ghosh

University of Kansas

April 29, 2019

Also Known As

- Profile- guided optimizations
- FDO
- PGO (sometimes called pogo)

OUTLINE

- Why ?
- How ?
- What happens when we do it ?

WHY ?

KAIZEN

KAI ZEN

改善

Change for Good

Static Analysis

Static Analysis

```
for( i=0; i<n; i++)
{
        bar();

}
```

Static Analysis

```
if (a<b)
        foo();
else
        bar();
```

Static Analysis

```
switch(i){
case 1:  …
case 2:  …

}
```

Static Analysis

```
If (a<b)
      foo(a);
else
      bar(a);
```

How frequently is a < b ?

```
switch(i){
case 1:  …
case 2:  …

}
```

Typical value of i ??

```
for( i=0; i<n; i++)
{
            bar();

}
```

How many times ??

Unanswered Questions

FDO

- Runtime optimization
- Using profile data for improvements

FDO

Aim: Smaller code + Faster execution

- Optimization based upon “how it behaves”
- “Hot sections” for speed
- “Cold sections” for size

HOW ?

STEPS

INSTRUMENT

INSTRUMENT

- “Building it again”
- Add “probes”

INSTRUMENT | TRAINING

- “Building it again”
- Add “probes”

INSTRUMENT | TRAINING

- “Building it again”
- Add “probes”
- Run test scenarios
- Dump profile data into files

INSTRUMENT | TRAINING | OPTIMIZE

- “Building it again”
- Add “probes”
- Run test scenarios
- Dump profile data into files

INSTRUMENT | TRAINING | OPTIMIZE

- “Building it again”
- Add “probes”
- Run test scenarios
- Dump profile data into files
- Reads profile data
- Optimizes code

PROBES

PROBES

Value Probes

Counter Probes

Static Analysis

```
if (a<b)
      foo(a);
else
      bar(a);
```

How frequently is a < b ?

```
switch(i){
case 1:  …
case 2:  …

}
```

Typical value of i ??

```
for( i=0; i<n; i++)
{
            bar();

}
```

How many times ??

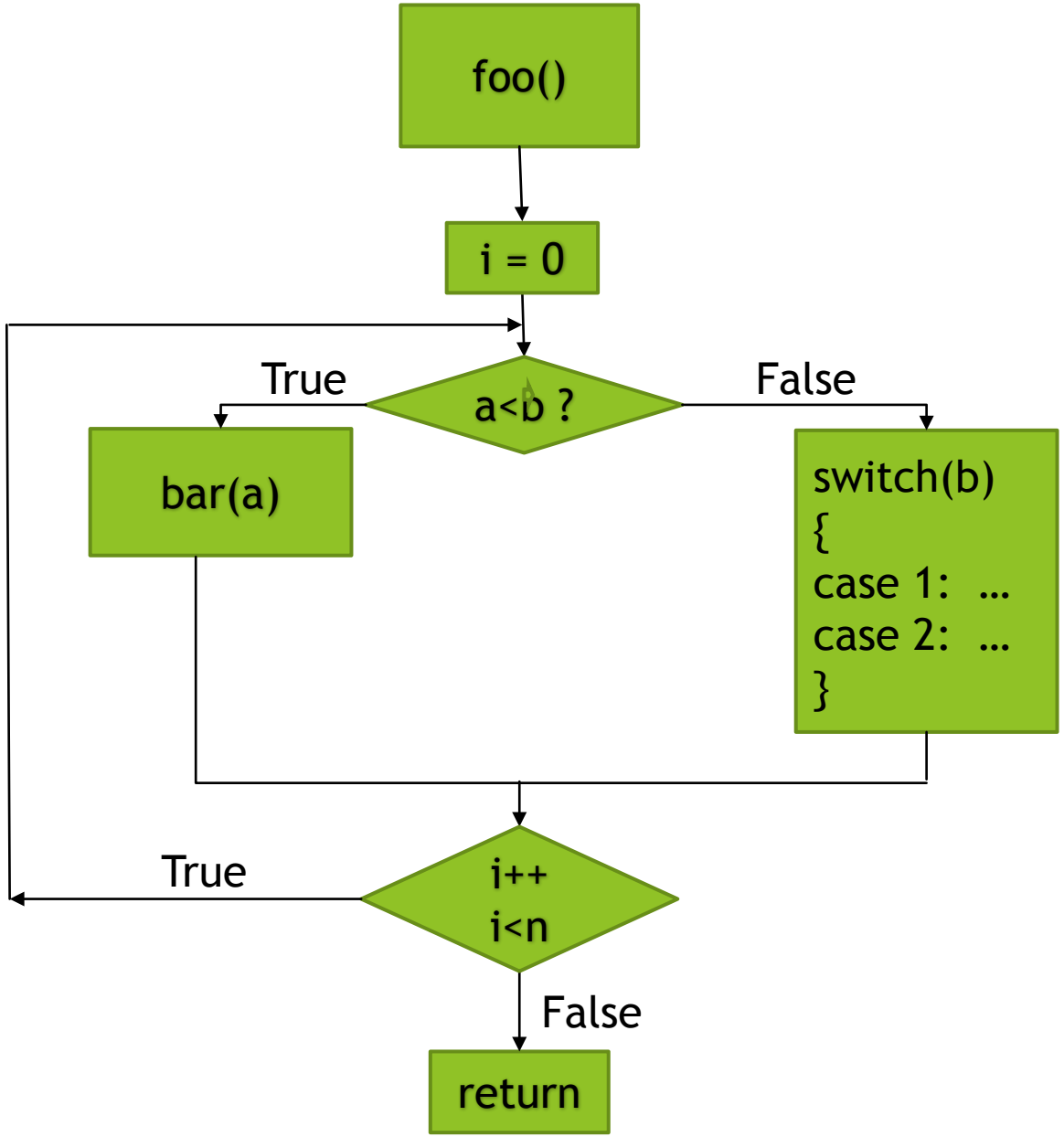
Unanswered Questions

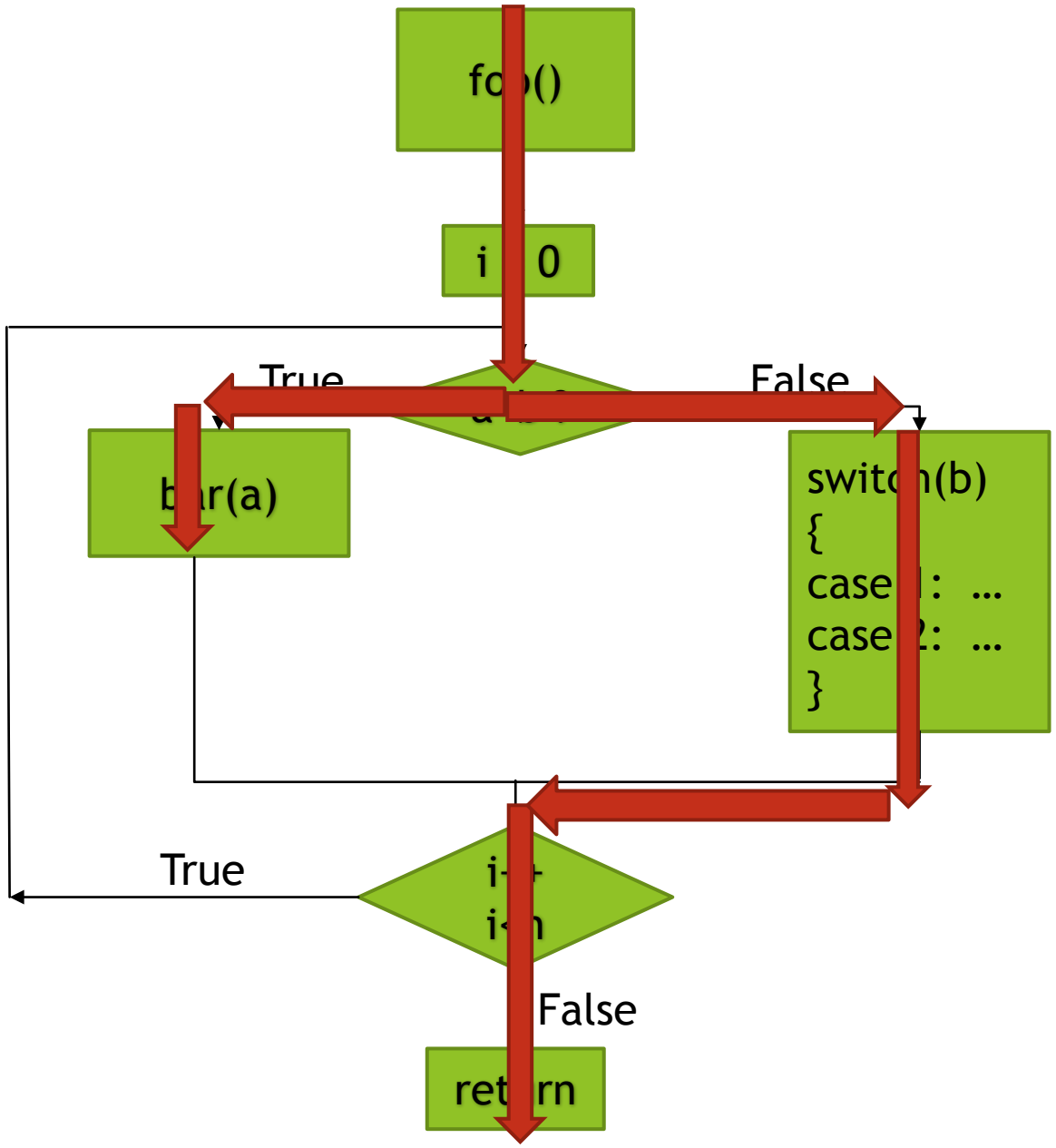
Example

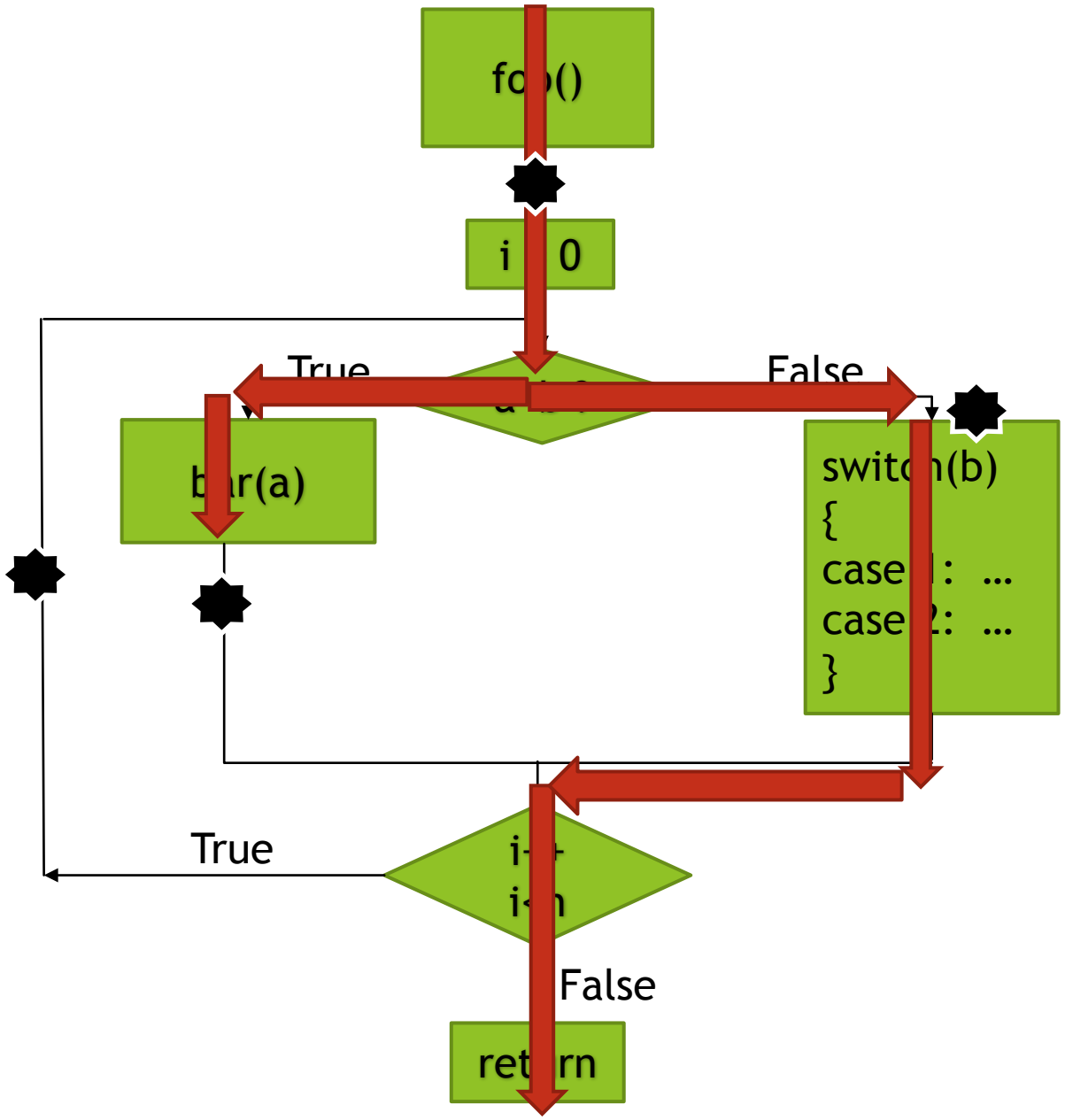
```
void foo(int a, int b)
{

int i;

for ( i=0; i<n; i++ )
      {
      if (a < b) {
                  bar(a);
                  }
      else
            switch (b) {
                        case 1:
                        case 2:
                  }
      }
      return;
}
```


foo()
i = 0
True
a<b ?
False
bar(a)
switch(b)
{
case 1: ...
case 2: ...
}
True
i++
i<n
False
return


fo()
i 0
True
False
b r(a)
switch(b)
{
case 1: ...
case 2: ...
}
True
i++
i<n
False
ret rn

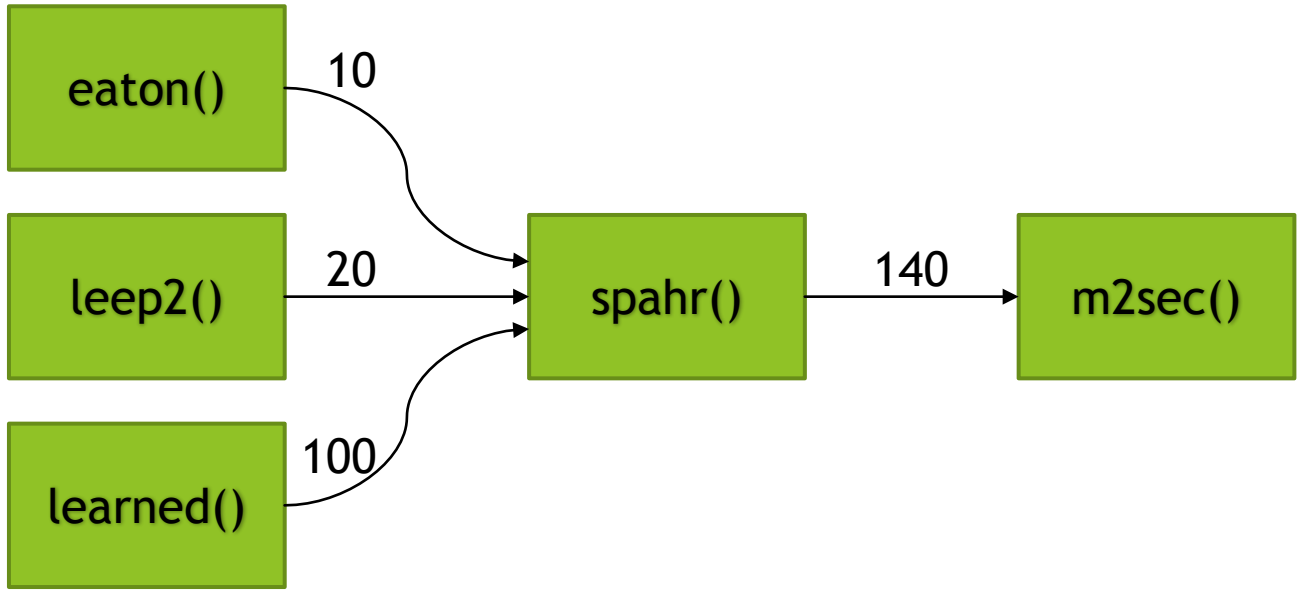

fo ()
i 0
True
False
b r(a)
switch(b)
{
case 1: ...
case 2: ...
}
True
i++
i<n
False
ret rn

OPTIMIZATIONS

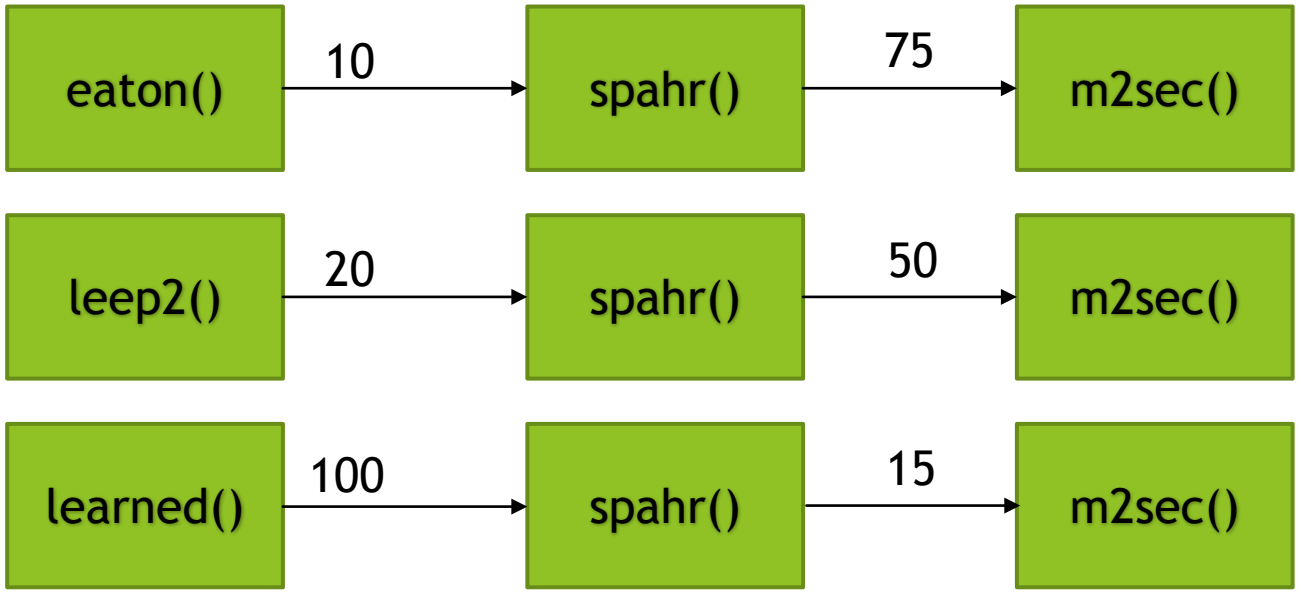
- Inlining
- Speed vs Size
- Block Layout
- Code Separation
- Function Layout
- Switch Expansion
- Partial Inlining

INLINING

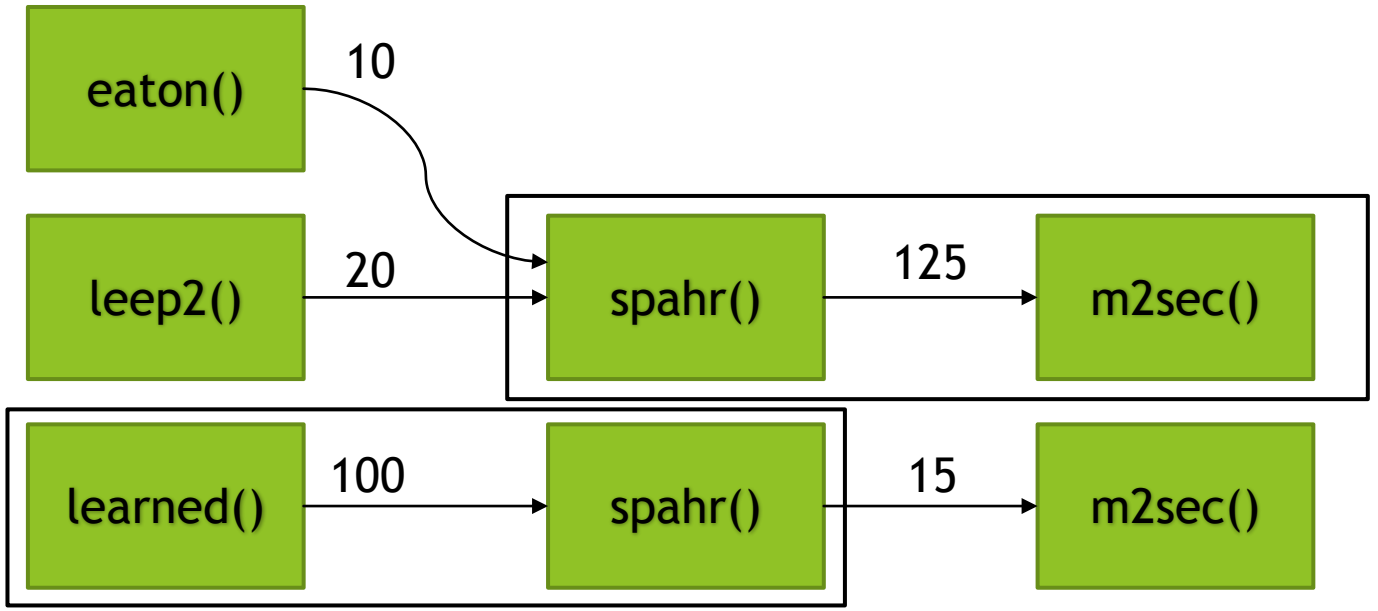
INLINING



INLINING

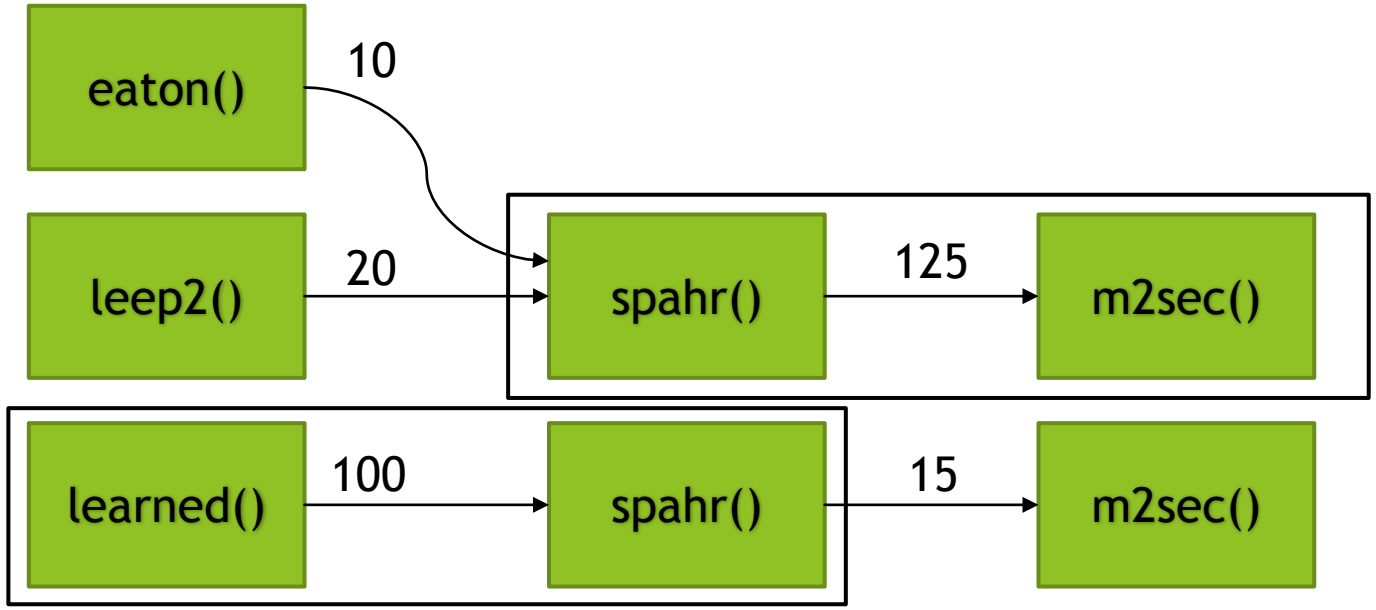

eaton()
10
spahr()
75
m2sec()
leep2()
20
spahr()
50
m2sec()
learned()
100
spahr()
15
m2sec()

INLINING



SPEED VS SIZE

SPEED VS SIZE

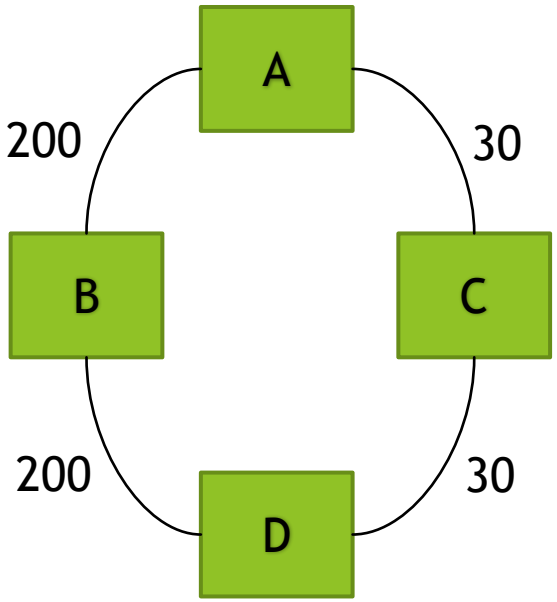


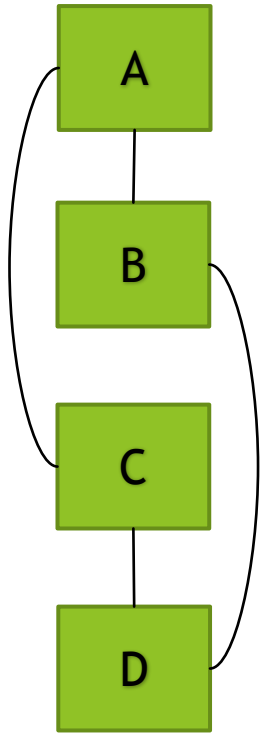
If **HIGH** Dynamic Instruction Count → SPEED
If **LOW** Dynamic Instruction Count → SIZE

Done after inline decisions are made

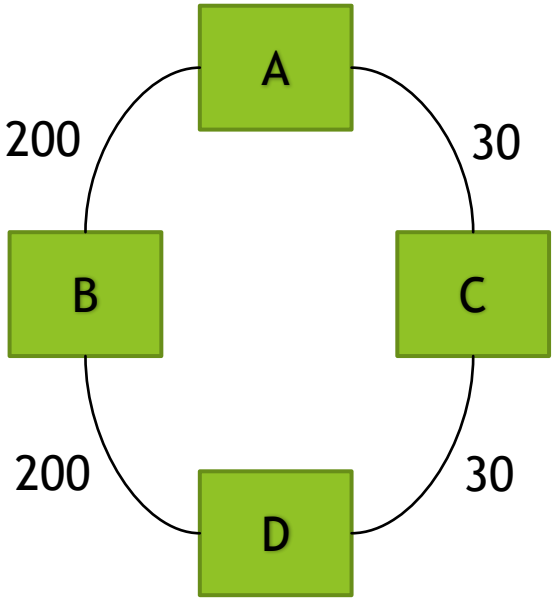
BLOCK LAYOUT

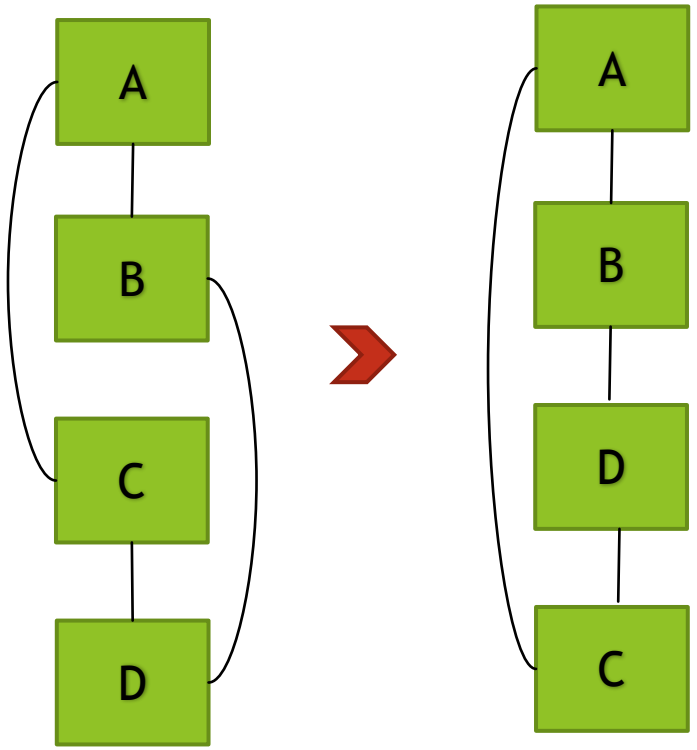
BLOCK LAYOUT





BLOCK LAYOUT

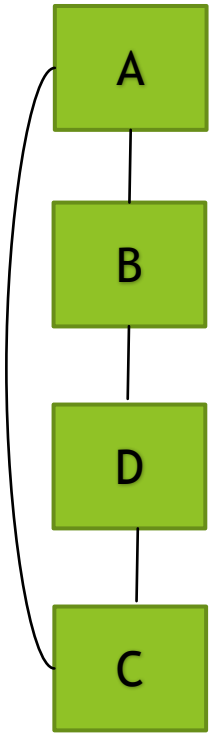




Improvement: Instruction cache locality

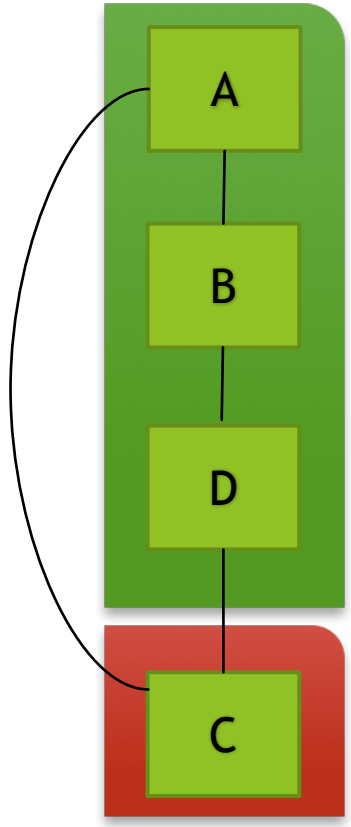
CODE SEPARATION

CODE SEPARATION



LIVE and **DEAD** code separated

CODE SEPARATION



Improvement: Better code locality, lesser working set

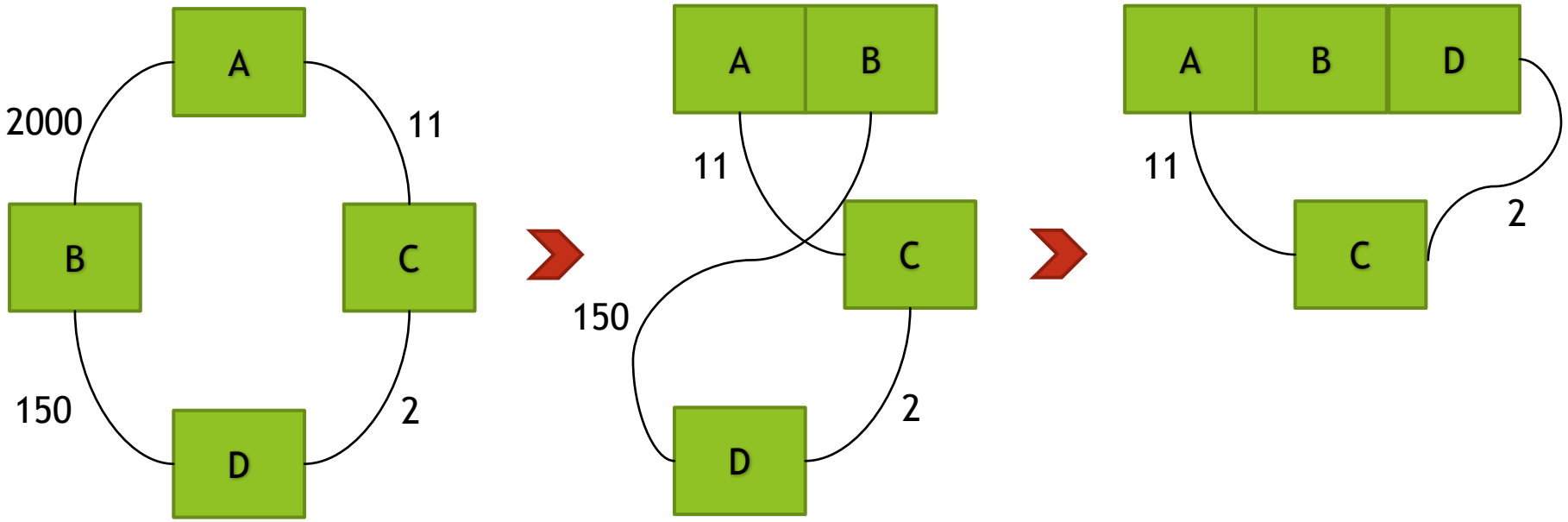
FUNCTION LAYOUT

FUNCTION LAYOUT

- Achieve page locality in functions
- Closest is best approach
- Connected functions are put together

Done after in-lining and code separation

FUNCTION LAYOUT



Done after in-lining and code separation

SWITCH EXPANSION

SWITCH EXPANSION

We know that
i = 5 most of the time

```
switch(i){
case 1:  …
case 2:  …

}
```

```
if( i == 5 )
      goto default;

switch(i){
case 1:  …
case 2:  …
default: …
}
```

PARTIAL INLINING

PARTIAL INLINING

Bar()

```
if ( a < b )
{
      a = a + b;
}
else Foo()
```

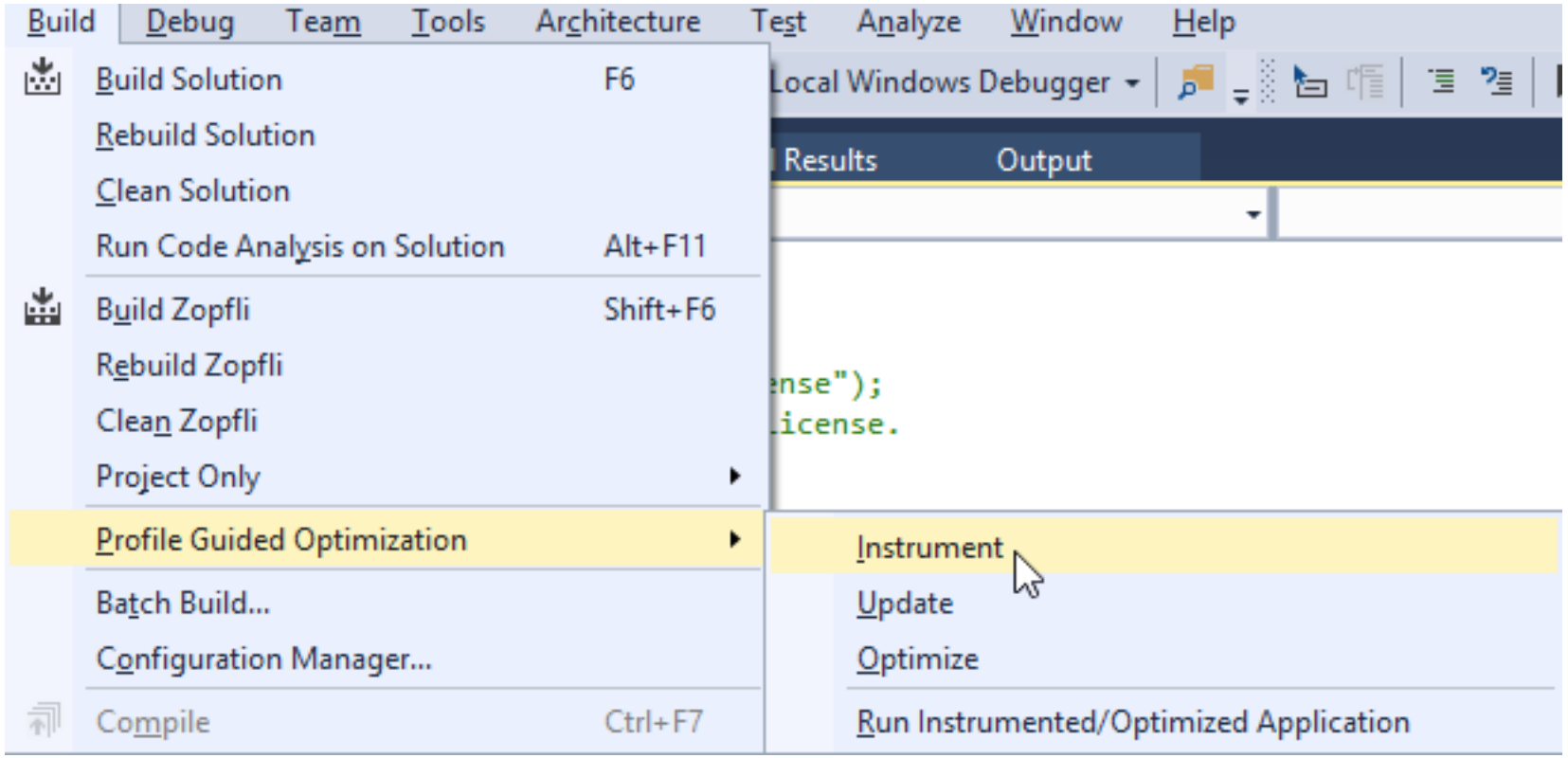
Hot section

Foo()

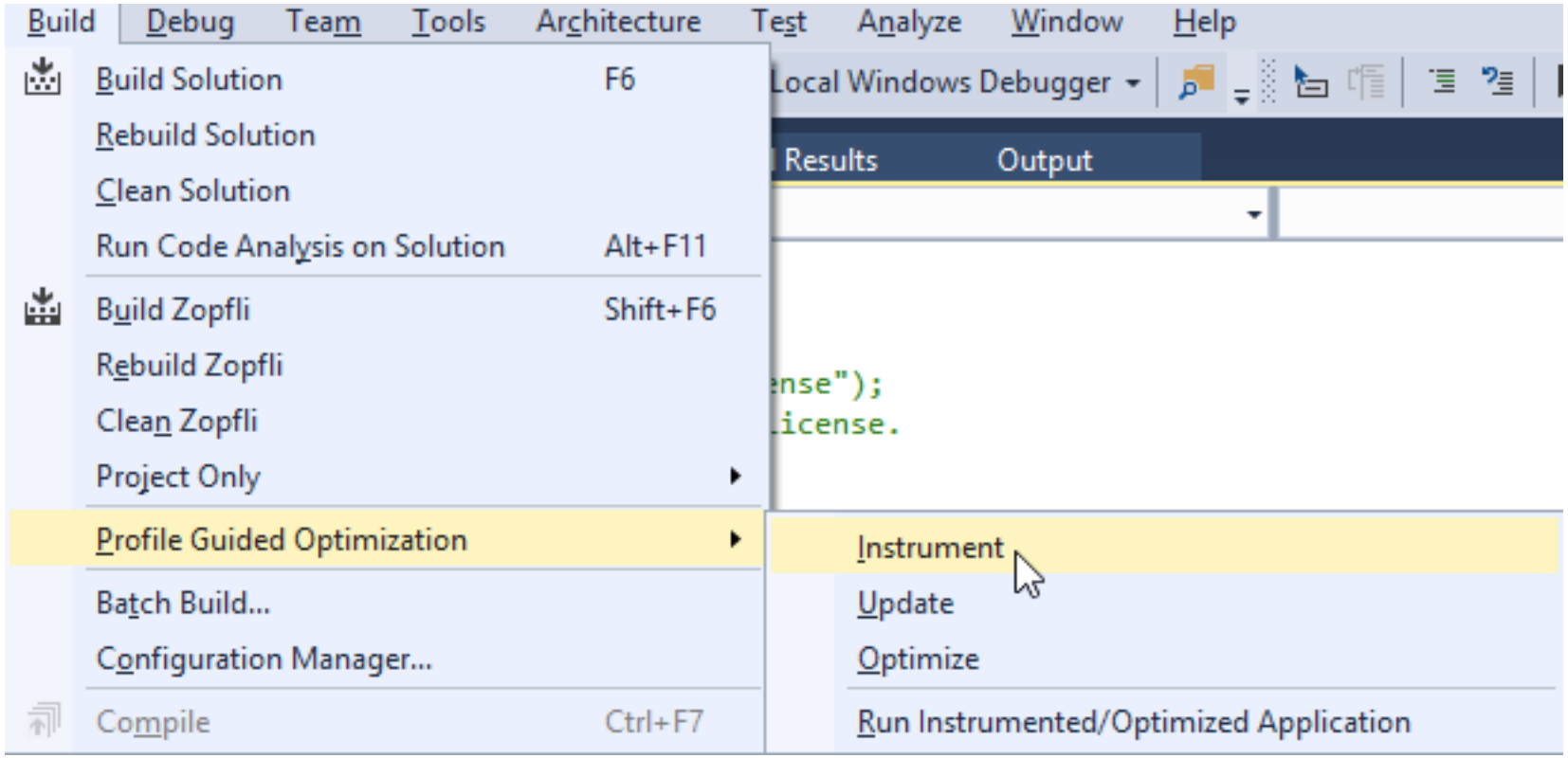
```
if ( a < b )
{
      a = a + b;
}
else
      .
      .
      .
      .
(never ending code)
```

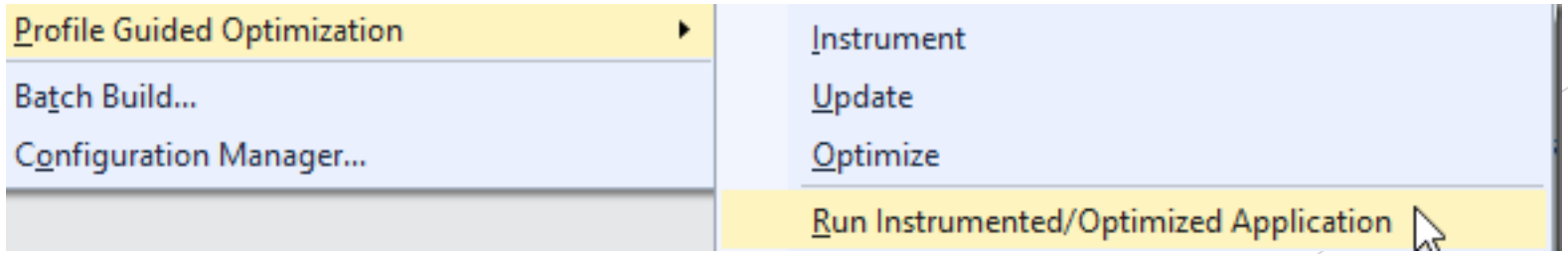
Improvement: Lesser page faults

VISUAL STUDIO

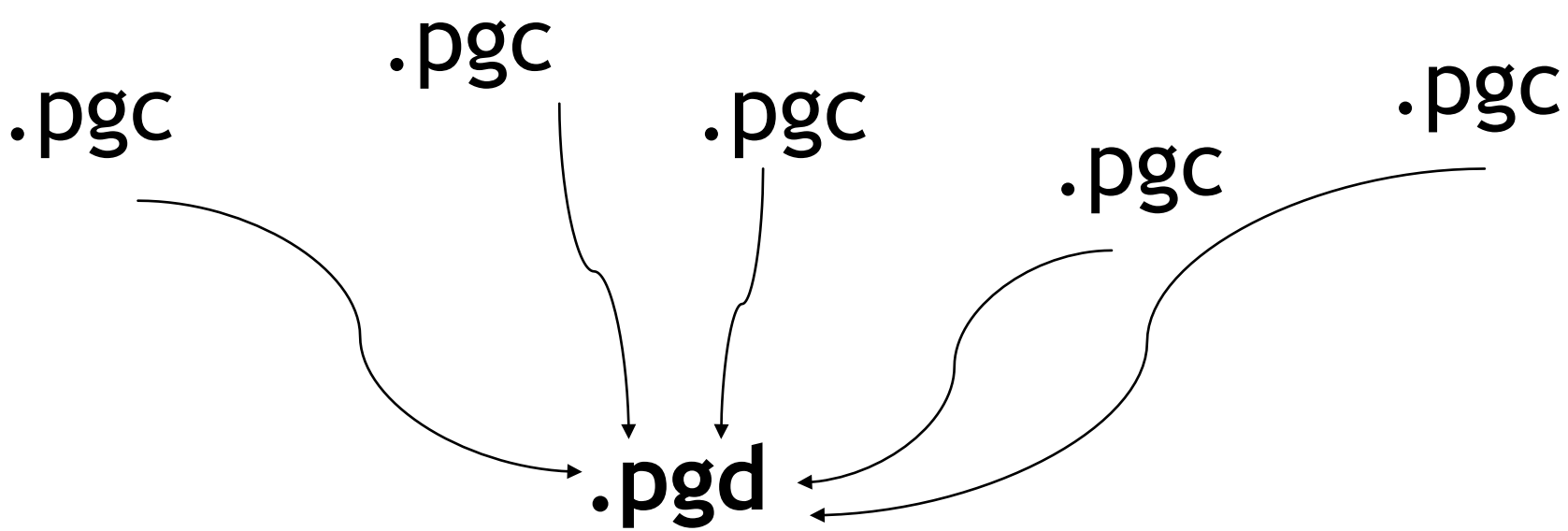


VISUAL STUDIO





VISUAL STUDIO



- In optimization stage, **pgd** is read
- Determine which functions will be compiled for speed/size
- Entry count, static/dynamic instructions count
- **pgomgr** tool
- Weights may be attached

WHAT HAPPENS ?

CASE STUDY

SPEC2K	Sjeng	Gobmk	Perl	POV-Ray	Gcc
LTCG size in MB	0.14	0.57	0.79	0.92	2.36
POGO size in MB	0.14	0.52	0.74	0.82	2.0
# of LTCG in-lines	163	2678	8050	9977	21898
# of POGO in-lines	235	938	1729	4976	3936
live section size (pogo)	0.5	0.3	0.25	0.17	0.77
% page locality	97	75	85	98	80
% speed gain	8.5	6.6	14.9	36.9	7.9


THE END
THE END
ANY QUESTIONS?
memecrunch.com

References

- Northwest C++ Users' Group: Ankit Asthana https://nwcpp.org/march-2013.html
- GoingNative :Program Manager Amit Mohindra and Development Lead Ten Tzen
- A Survey of Adaptive Optimization in Virtual Machines: MATTHEW ARNOLD, STEPHEN J. FINK, DAVID GROVE, MICHAEL HIND, AND PETER F. SWEENEY, MEMBER, IEEE