

# A Taxonomy of Programmer Activities for Studying Parallel Programming in the Small

Jaymie Strecker

UMD – HPCS

June 15, 2005

# Why study programmers' activities?

- Evaluate language features w.r.t. effort
- Evaluate tools w.r.t. effort
- Discover problems new tools could address
- Discover effort-wasting bug classes
- Understand cost-benefit curve for program optimization
- Understand “lone researcher” context
- Discover patterns in programmers' activities
- .....

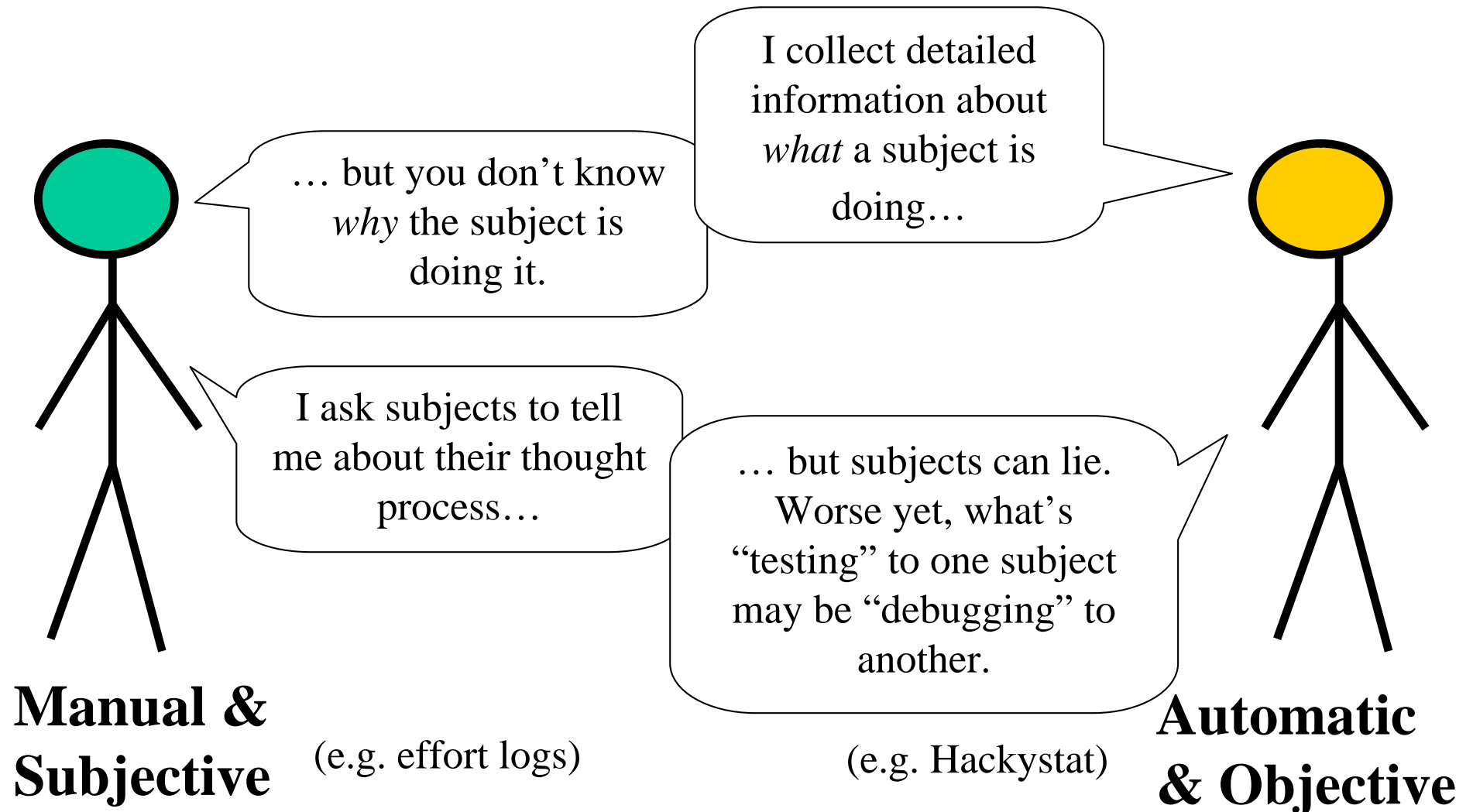
# How We Track Activities

## Activity-tracking mechanisms

Manual reporting by subjects	Manual-automatic hybrid	Automatic reporting by instrumentation
Effort logs	Interactive compiler wrapper	Wrappers around compiler and job scheduler, Hackystat

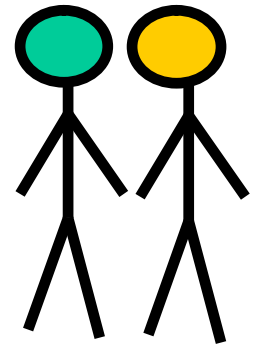
## Taxonomy of activities

# Dichotomy of Activity Tracking



# Manual/Subjective + Automatic/Objective Reporting

- Ideal:
  - Combine advantages of both styles
    - Track what subject is doing *and* why
    - Check agreement between manually and automatically collected data
- A first step:
  - Instrumented compiler
    - *Automatically* triggered when subject invokes compiler
    - Asks subject for *manual* response



# Instrumented Compiler

Reason for recompile:

- (1) Serial coding
- (2) Parallelizing the code
- (3) Testing
- (4) Debugging
- (5) Tuning
- (6) Experimenting with environment
- (7) Other (you will be asked to supply reason

>

# Ambiguous Activity #1

```
a = b*c;
```

```
printf("a=%d, b=%d, c=%d", a, b, c);
```

I know 'a' is wrong.  
Where's the bug?



Debugging

I think 'a' is right, but  
I want to check it



Testing

# Ambiguous Activity #2

```
void do_stuff() {  
    ... /* changes made here */  
}
```

```
/* start parallel work */  
...  
do_stuff();  
...  
/* end parallel work */
```

**Serial or parallel?**

# Ambiguous Activity #3

Write some parallel code...

Rewrite some serial code...

Write more parallel code...

Optimize code...

Write documentation...

Inspect code...

Fix a mistake...

**Compile!**



# The Big Picture

- What do we want to learn, anyway?
- One goal:
  - *G1: Characterize the steps a programmer takes in developing a parallel program with respect to the type of work and the effort required.*

# GQM Example: Timed Markov Workflow Model

**G1:** Characterize the steps a programmer takes in developing a parallel program with respect to the type of work and the effort required

**Q1:** How well can a timed Markov model based on a programmer's past activities predict the effort the programmer spends on future work?

**M1:** sequence of activities (nodes)

**M2:** time spent in each node

**Taxonomy of activities**

```
graph TD; G1[G1: Characterize the steps a programmer takes in developing a parallel program with respect to the type of work and the effort required] --- Q1[Q1: How well can a timed Markov model based on a programmer's past activities predict the effort the programmer spends on future work?]; Q1 --- M1[M1: sequence of activities (nodes)]; Q1 --- M2[M2: time spent in each node]; M1 --> Tax[Taxonomy of activities]; M2 --> Tax;
```

# Rebuilding the Taxonomy

- Let's rebuild the taxonomy of activities so that it's easier to measure M1 (sequence of activities) and M2 (time spent in each).
- Start with no-brainers...
  - “Improve clarity/design” activity
    - e.g. commenting code
  - “Transfer code” activity
    - red flag for missing data

# Rebuilding the Taxonomy

- Recall Ambiguous Activity #1
  - hard to distinguish “testing” from “debugging”
- Replace “test” and “debug” with...
  - “repair” – trying to repair a bug
    - editing code to improve correctness
  - “diagnose” – trying to find a bug, or to determine that no bugs exist
    - editing code to aid diagnosis (e.g. print statements)
    - other non-editing actions

# Rebuilding the Taxonomy

- Recall Ambiguous Activity #2
  - hard to distinguish “serial coding” from “parallelizing the code”
  - what about “serial debugging” vs. “parallel debugging”?
- Replace “serial coding” and “parallelizing” activities with
  - “add functionality”
- Instead of serial/parallel *coding*, think of serial/parallel *code*
  - parse programs into components

# Program Components

<b>Serial program</b>
Do work

<b>Parallel program</b>
Pre-parallel work (e.g. process command line arguments)
Initialize communications/multi-threading
Divide work between processors
Do work in parallel
Combine results from processors
Finish communications/multi-threading

Post-parallel work (e.g. print result)

# Structure of Rebuilt Taxonomy

Let's *very specifically* define each activity in the taxonomy to prevent overlaps.

Programmer's behavior described by 2 layers:

- Type of work (*more about this in a moment*)
- Part of program being worked on

For example:

- Type of work = edit code to improve speed
- Part of program = initialize communications/multi-threading

# Types of Work

Activity	“AFK” Work	Code-Changing Work	Code-Running Work
Diagnose	Read/inspect code	Edit “white box” code	Run program as “white box” Run program as “black box” Use a debugging tool

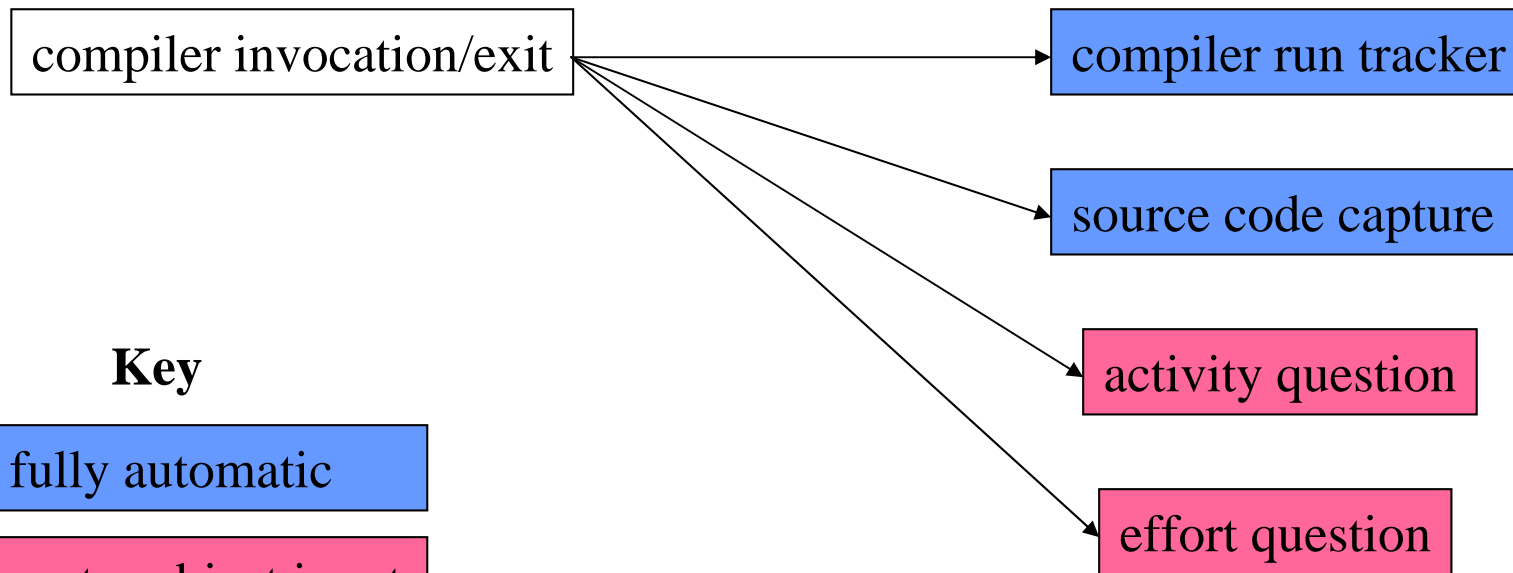
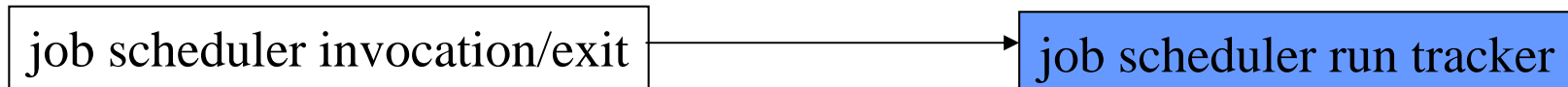
# Rebuilding the Instrumentation

- Recall Ambiguous Activity #3
  - subjects can do multiple activities between compiles
- An issue with the instrumentation, not the taxonomy
- So let's generalize the instrumented compiler.

# Current Instrumentation

## Event Types

## Event Responses



## Key

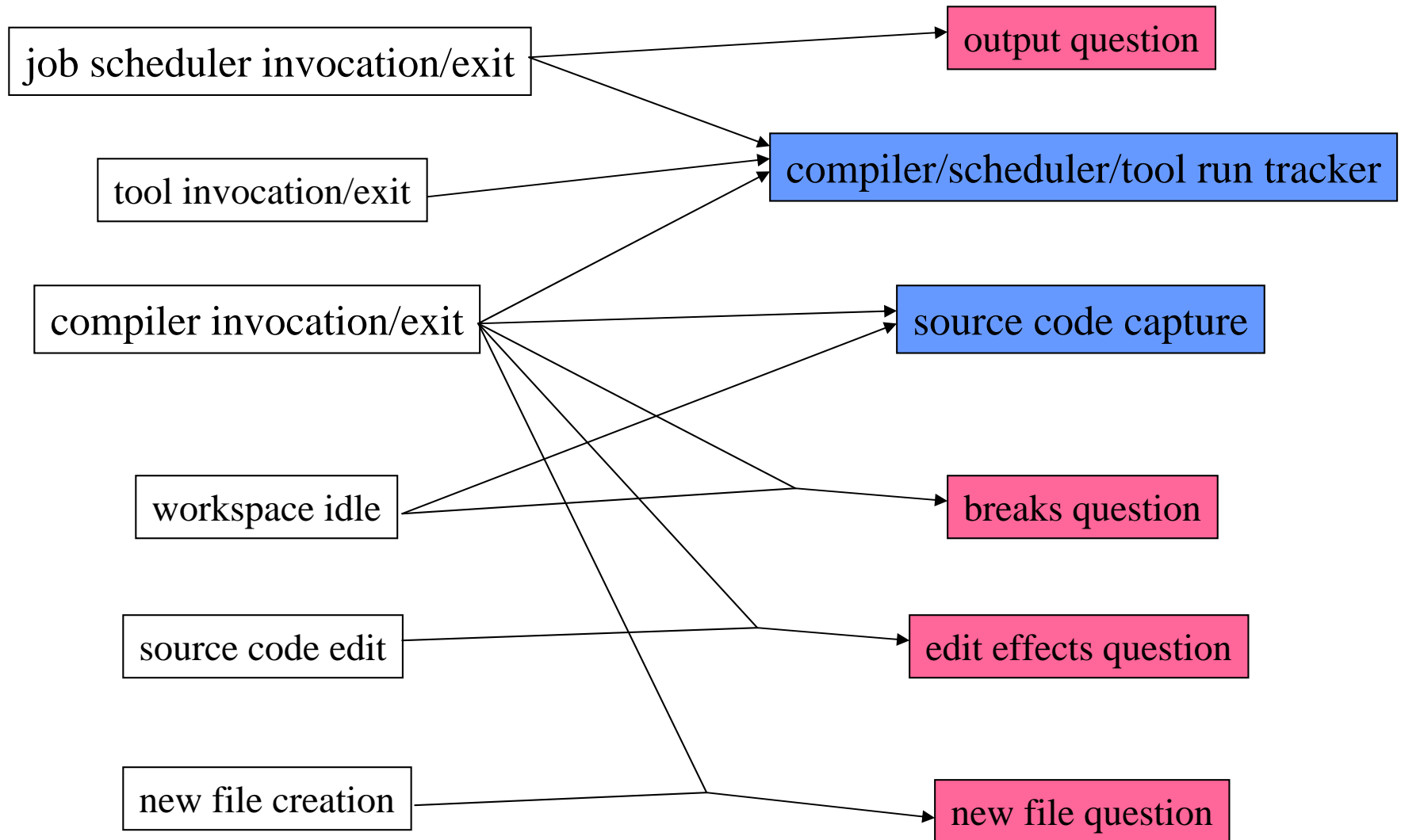
fully automatic

requests subject input

# Rebuilt Instrumentation (Envisioned)

## Event Types

## Event Responses



# Rebuilt Instrumentation - Key Points

- Capture data at more of the programmer's stopping points
  - invocation/exit of compiler, job scheduler, tools
- Use automatically collected data in real time to direct interactive collection
  - **Hackystat** tracks active time and edited files
- Ask subject more, but easier, questions
  - “Were you debugging, testing, ...?” becomes “Was the output of your program correct?”

# Questions for the Subject

output question

Was the output of this run correct?

breaks question

Your workspace was idle for 50 minutes since 4:45 p.m. How much of that time was NOT spent working on your program?

edit effects question

What effects might this change have on your program?

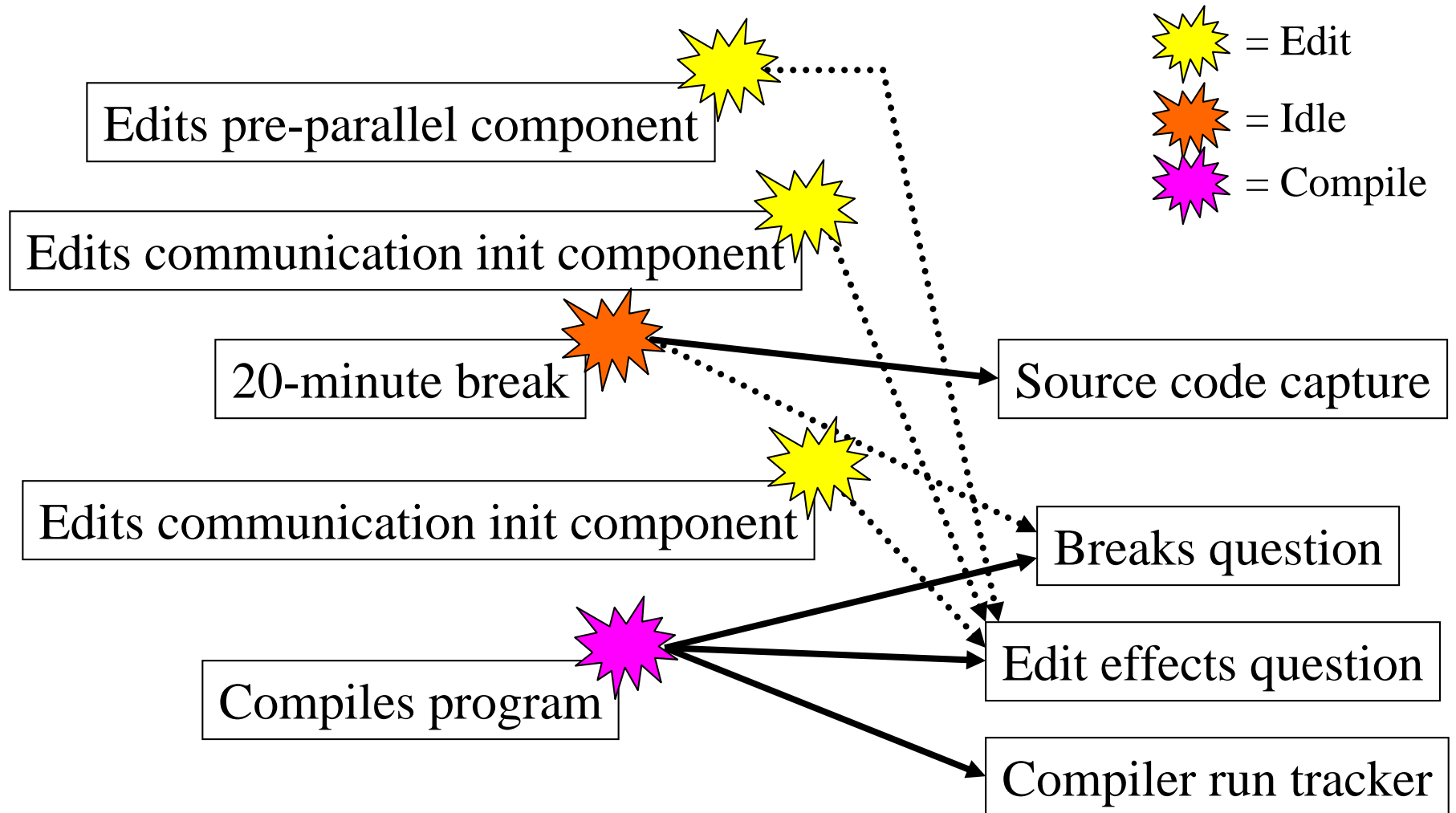
- (a) Improve correctness
- (b) ...

new file question

How did you begin the new file *foo.c*?

- (a) Transferred from another computer
- (b) ...

# Example of Instrumentation Use

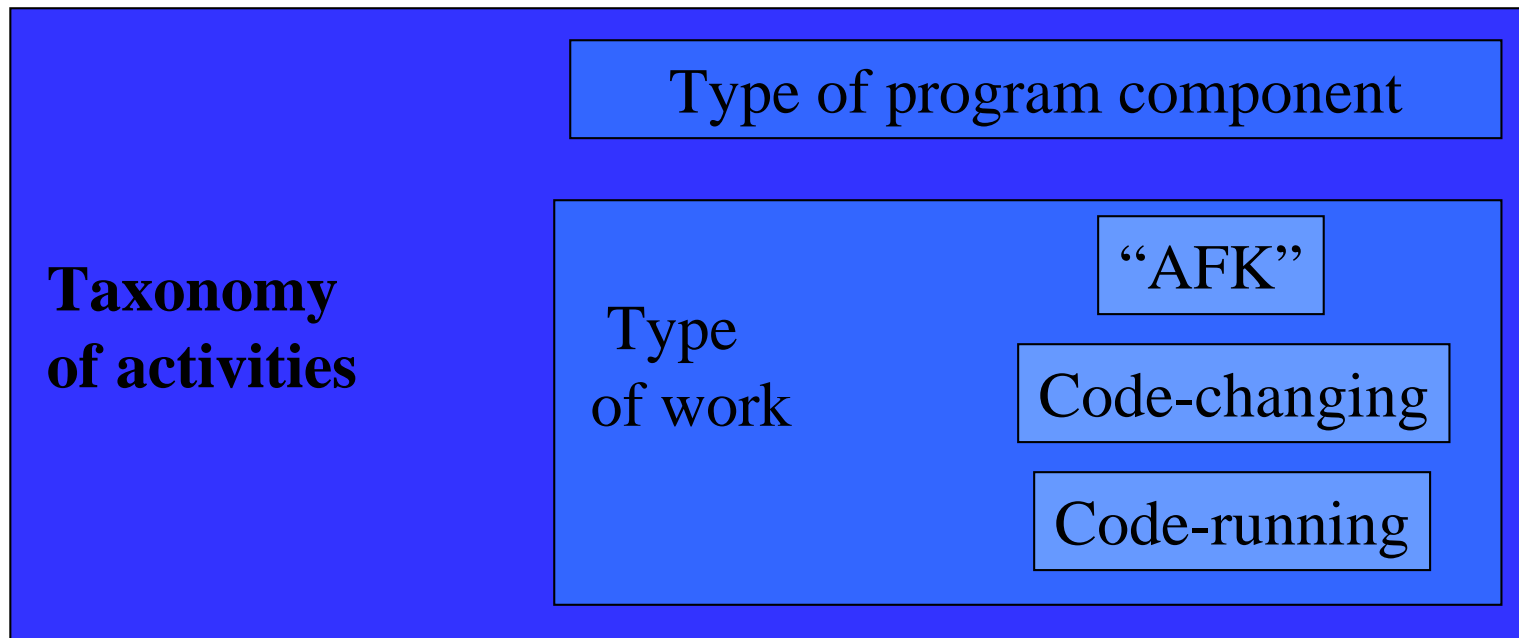


# Evaluation of Instrumentation

How many bad answers does the subject give to the interactive instrumentation?

- Upper bound
  - observational studies
- Lower bound estimate
  - assume subject always answers randomly
  - fewer answer choices => better accuracy
- Practical estimate
  - check in on subjects occasionally (instant messenger?)
  - compare to manual effort logs

# Summary



Questions?

This document was created with Win2PDF available at <http://www.daneprairie.com>.  
The unregistered version of Win2PDF is for evaluation or non-commercial use only.