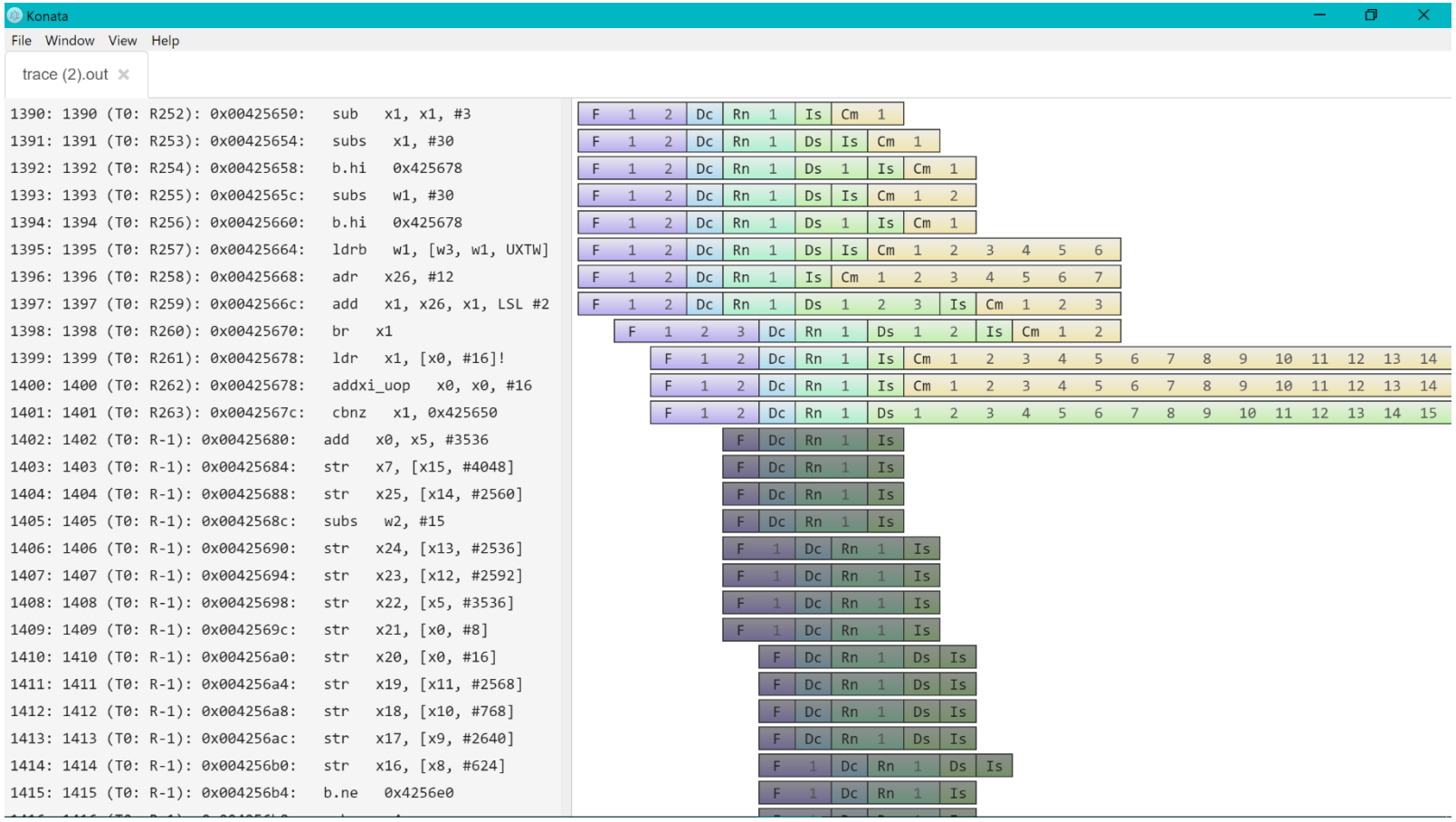
Visualizing the out-of-order CPU model

Ryota Shioya
Nagoya University

Introduction

- This presentation introduces
  - the visualization of the out-of-order CPU model in gem5



Introduction

- Let's suppose
  - you come up with an excellent idea and
  - try to extend the CPU model in gem5 for adding your new method.
- You will probably tackle the following issues:
  - difficult bugs, especially performance related ones
  - a situation where your method cannot improve the performance as expected

Introduction

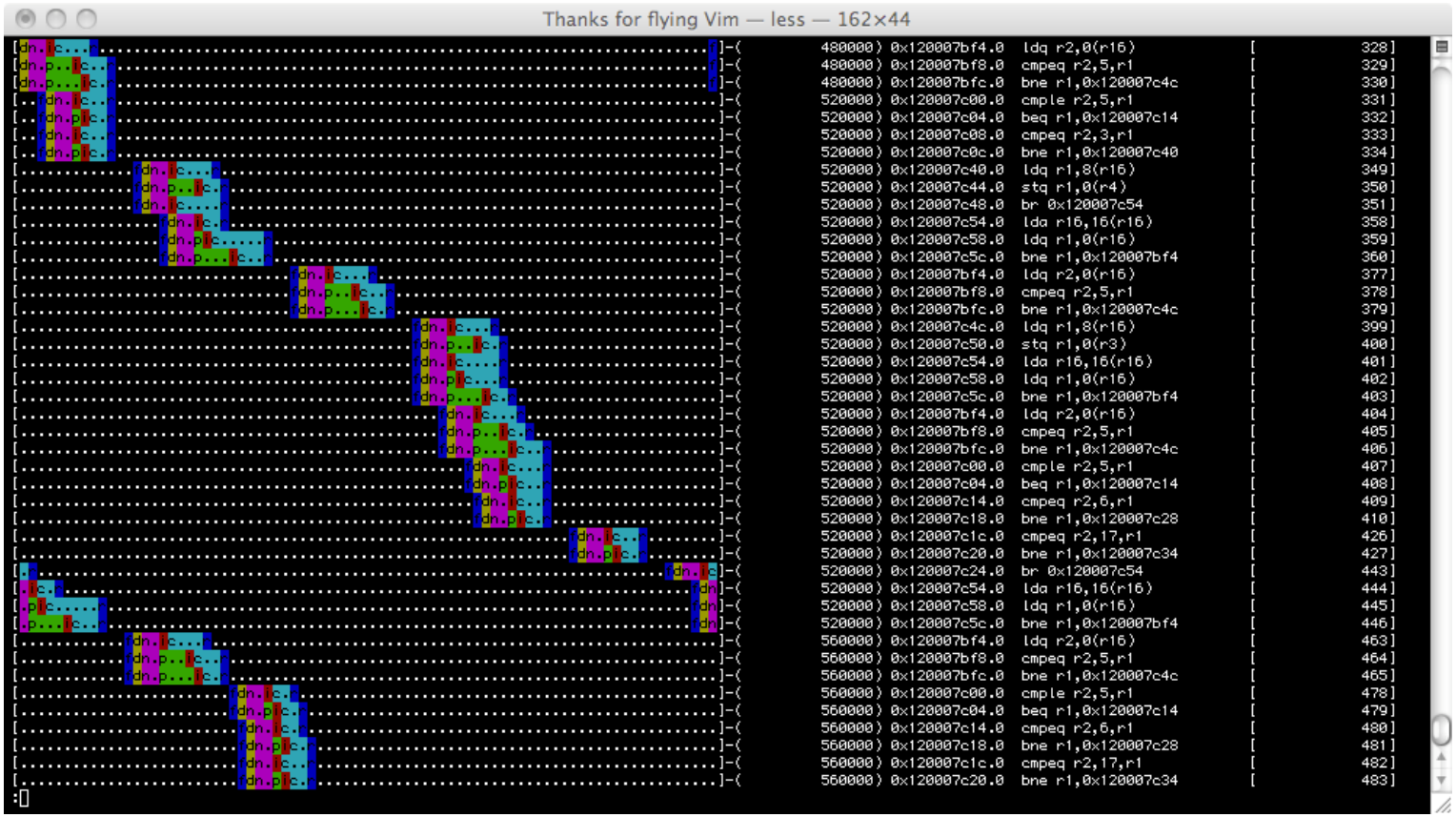
- You probably investigate your modified gem5 as follows:
  - Check counters outputted by gem5
    - e.g. the number of LLC misses / branch mispredictions
    - These counters sometimes give us clues.
  - Check the behavior by using a debugger and step execution
- But, it is difficult to fix issues in the following situations:
  - You have no idea what causes it
  - You recognize some counters show that something is wrong, but you have no idea what happened

Visualizing the pipeline behavior

- In such situations, pipeline visualization is very useful.
- In general, visualization is a powerful tool for investigating bugs or behavior.
  - If you have developed hardware with HDL such as Verilog, you may have used a waveform viewer.
    - In a waveform view, you can easily see signal transitions and relations between signals.
    - Such viewers may have helped you a lot.
- This is also true for gem5!

A text-based pipeline viewer is provided for gem5

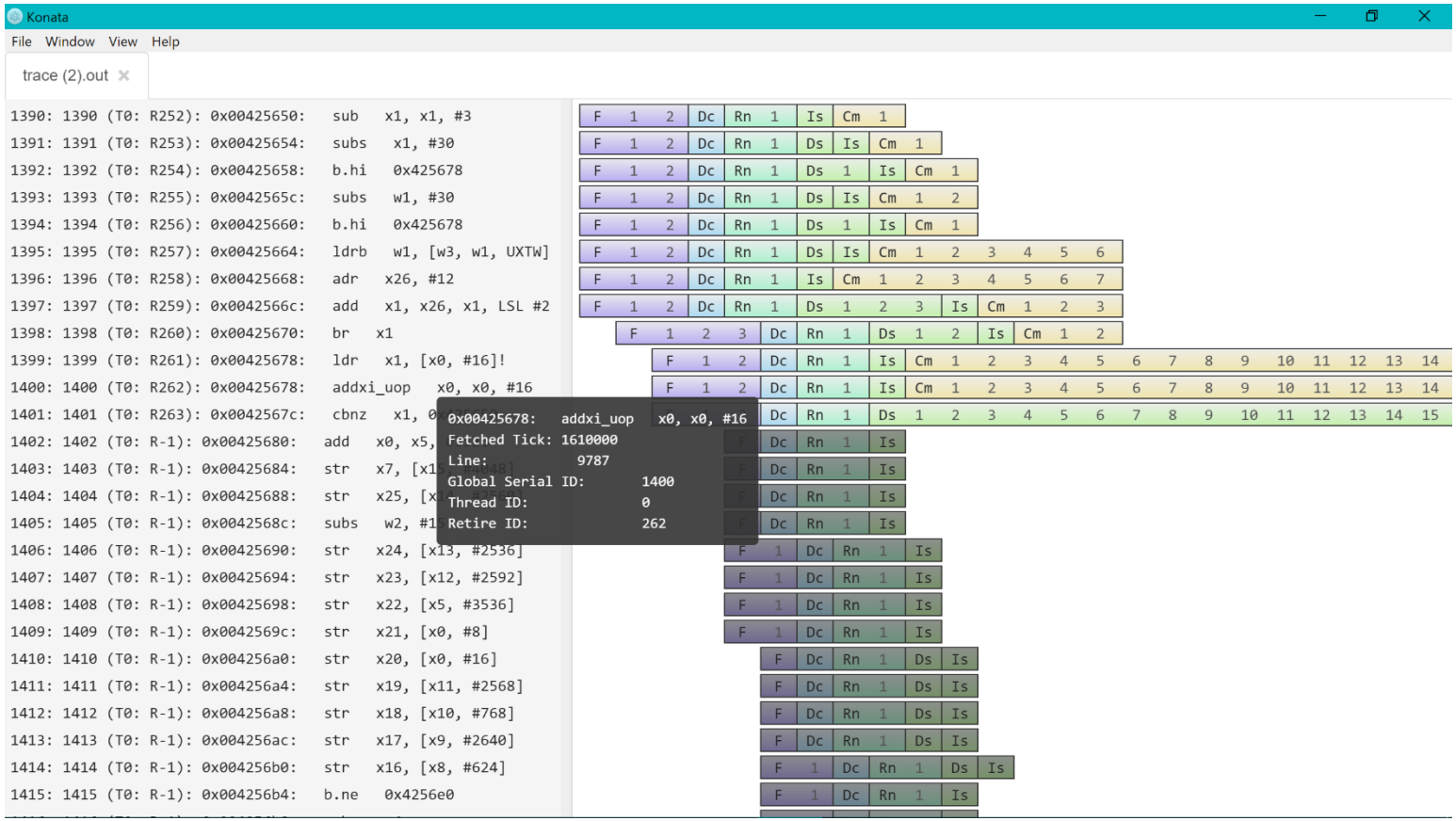
- This viewer is very useful to investigate the pipeline behavior.
  - But, you can see only a limited range of instruction sequences at once
  - This is the "less" command itself, it is not very user-friendly.



This picture is from http://www.m5sim.org/Visualization

Konata: a new GUI based viewer

- You can see the pipeline behavior as a map app.
  - This presentation introduces Konata and best practice in gem5.



Outline

1. A brief explanation of how to use
2. Typical visualization examples
3. Use cases

Preparation

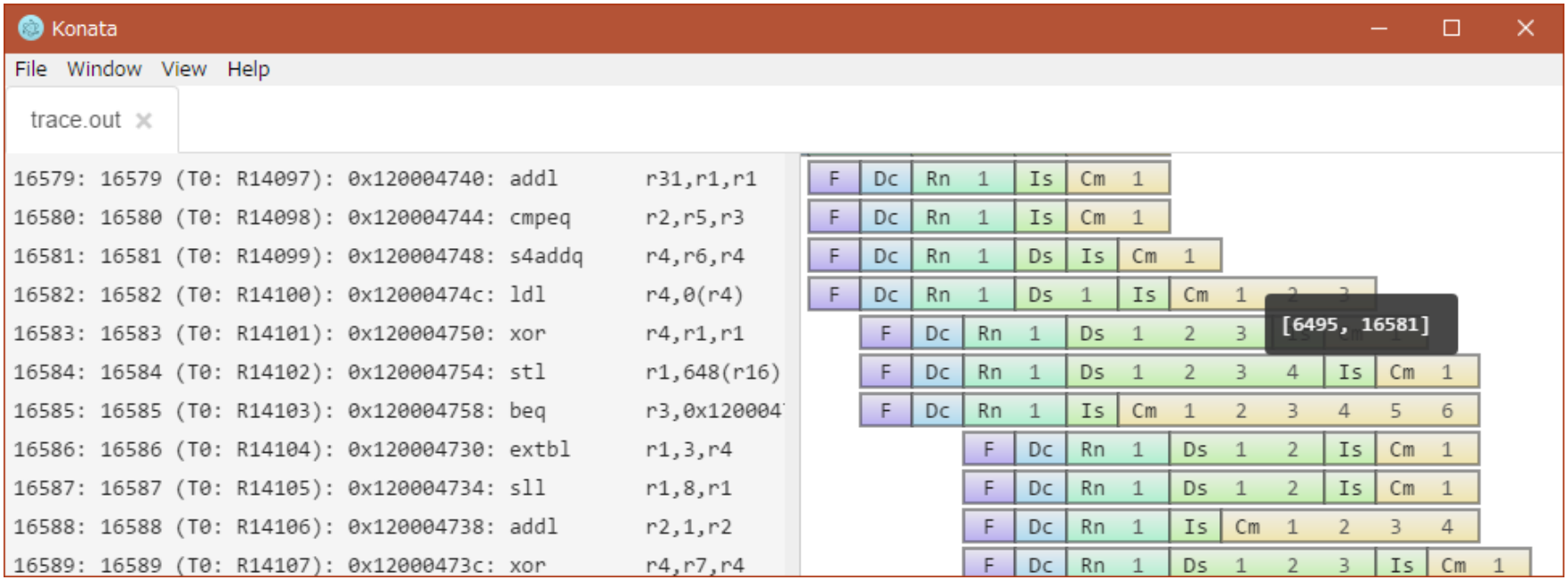
1. Install: All you have to do is to download the package and unpack it.
   - https://github.com/shioyadan/Konata/releases
   - Windows/Linux/Mac packages are provided.
   - No additional runtime is not required
2. Start the executable file such as Konata.exe

How to Use

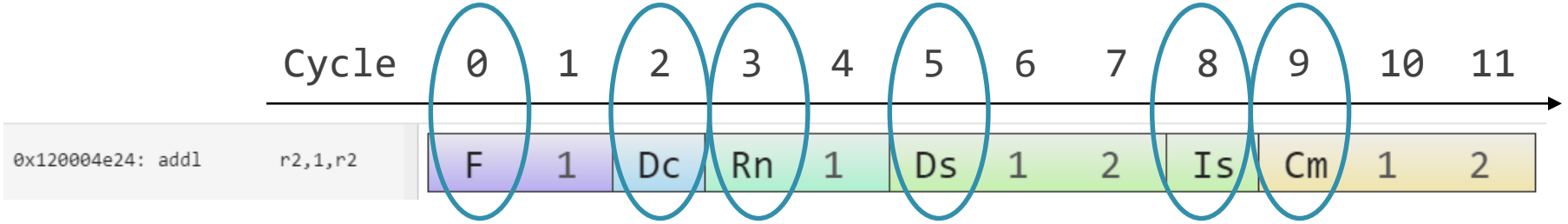
1. Generate a trace log from gem5 with the O3 CPU model
   - Execute gem5 with the following flags
   - `./build/ARM/gem5.opt --debug-flags=O3PipeView --debug-start=<first tick of interest> --debug-file=trace.out configs/example/se.py --cpu-type=detailed --caches -c <path to binary> -m <last cycle of interest>`
   - This example is from http://www.m5sim.org/Visualization
2. Load the generated "trace.out" to Konata
   - from the menu in the window or using drag&drop

How to use

- After loading the file, contents like the following are shown.
  - Left side: instruction information such as a PC and mnemonic
  - Right side: the image of visualized pipeline behavior



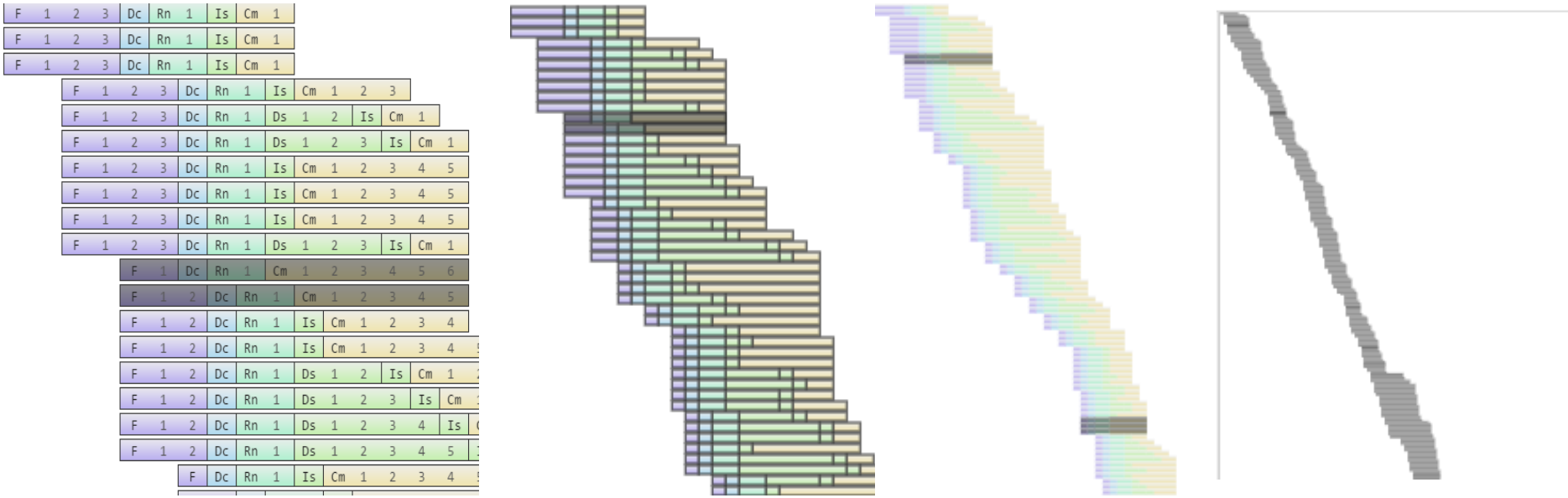
How to see the visualized image



- The clock cycle proceeds from left to right
  - F : Instruction fetch
    - In this insn., the fetch latency is 2 cycles
  - Dc : Instruction Decode
  - Rn : Rename
  - Ds : Dispatch
  - IS : Issue
  - Cm : Completion of execution
    - The execution stage is not explicitly shown
  - (The end of Cm stages) : Retire

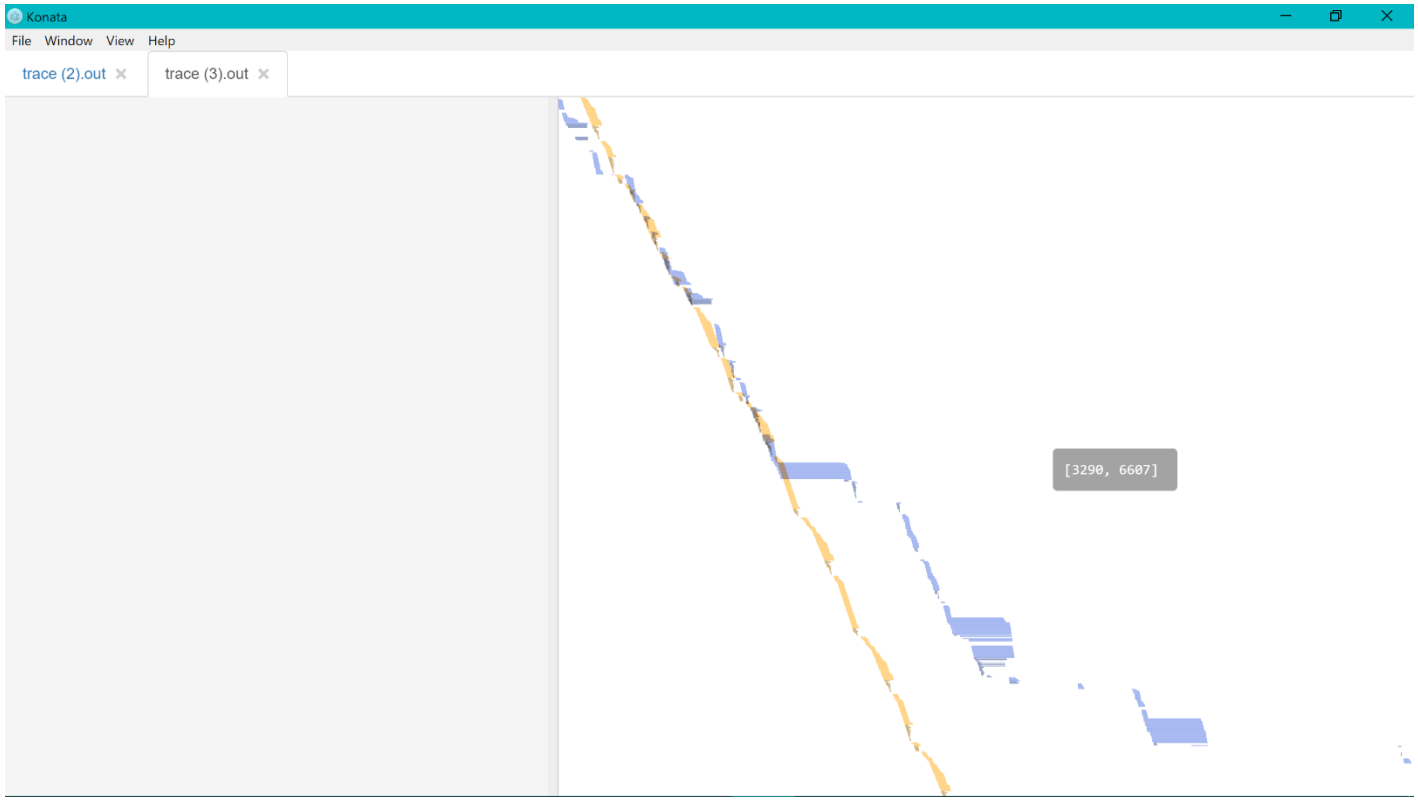
Zoom in/out

- You can zoom in/out as follows:



Compare Two Pipelines

- Konata can show two pipelines overlapping as follows:
  1. Load two files
  2. Right click -> "Transparent mode" & "Synchronize scroll"

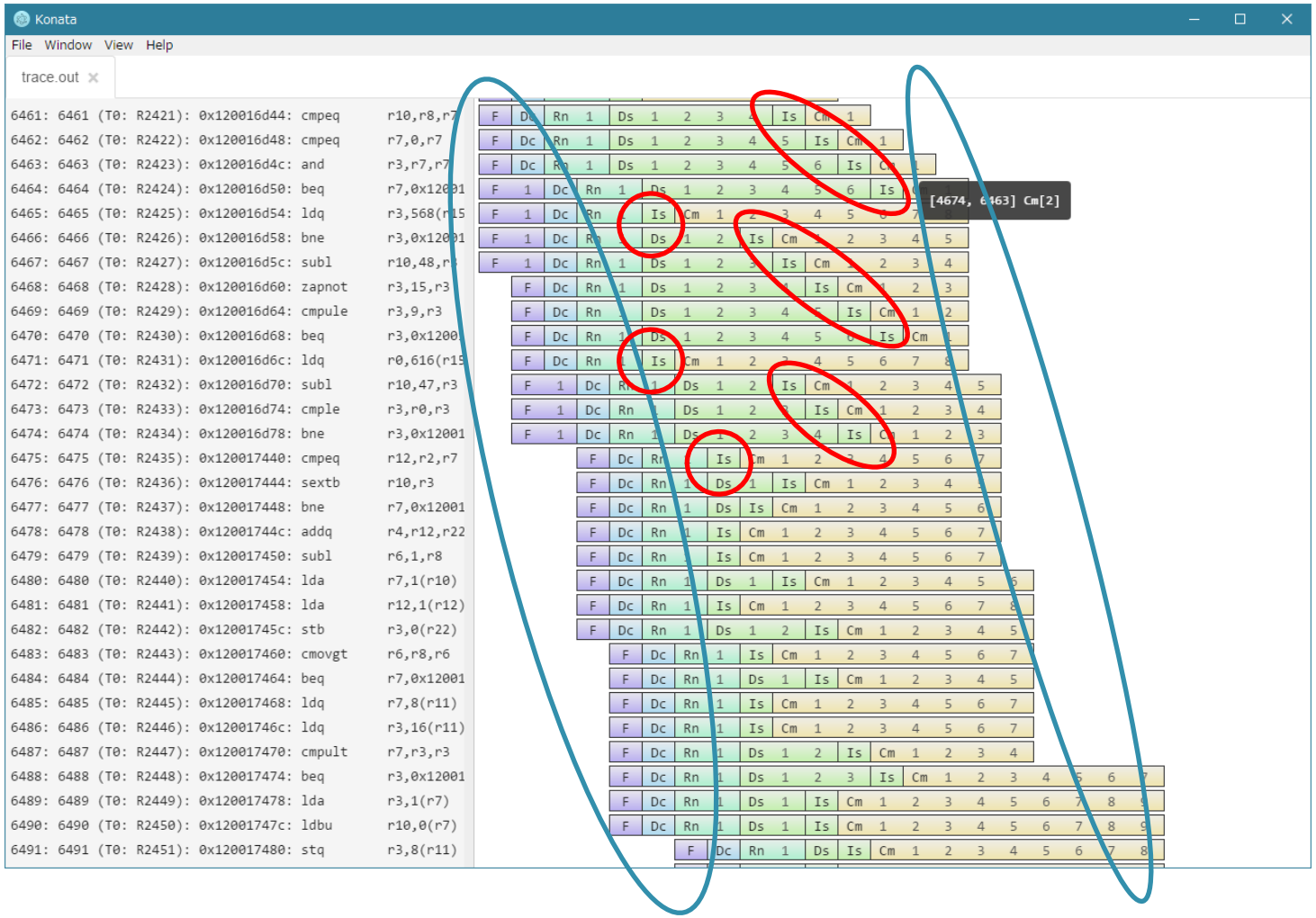


Typical Visualization Examples

- Introduce how the following things are shown:
  1. Out-of-order execution
  2. Branch misprediction
  3. Cache miss
  4. Execution speed

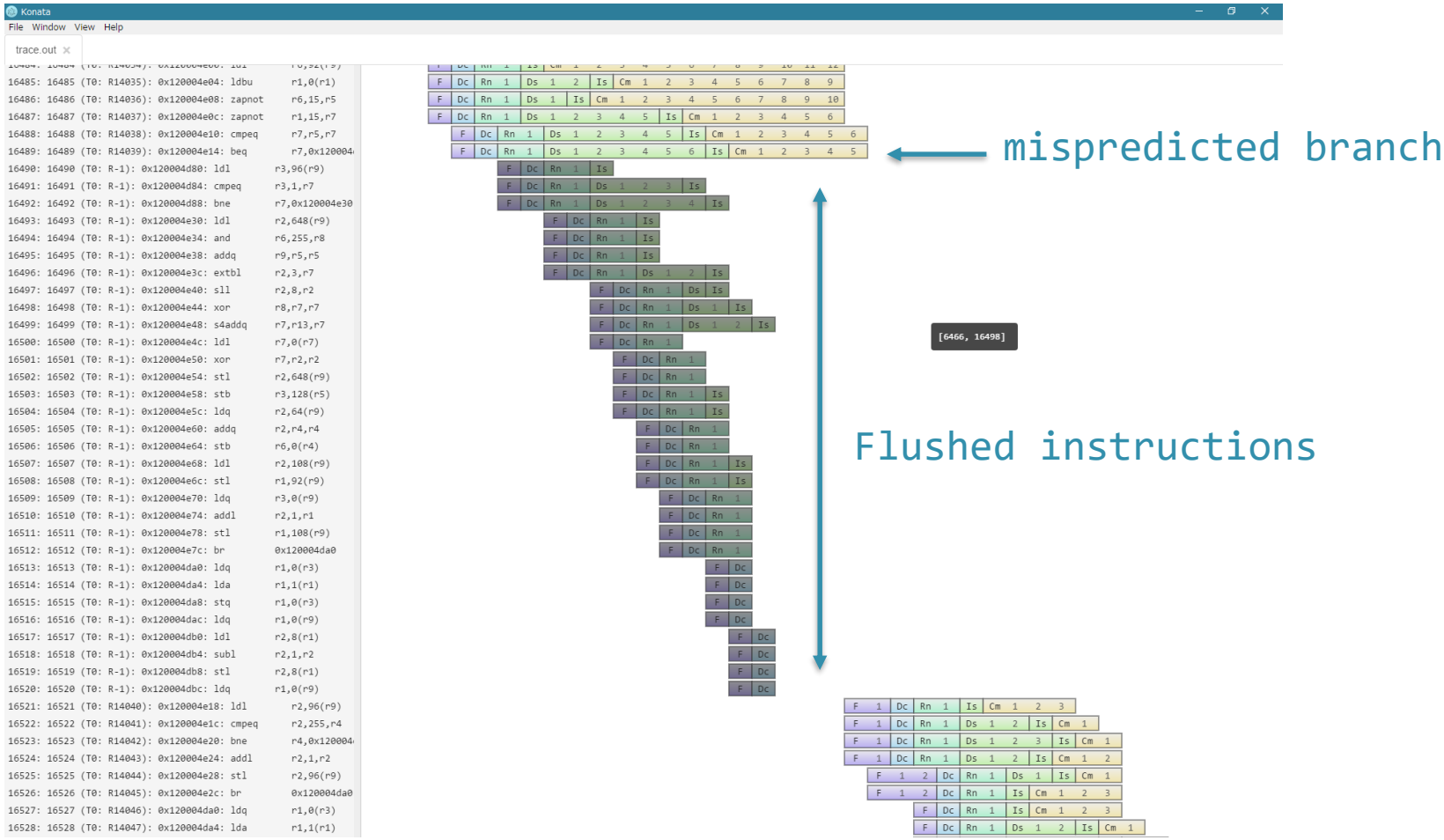
Example: Out-of-order Execution

- Fetch and retirement, marked with the blue circles, are performed in-order
  - Instruction issue, marked with the red circles, is performed out-of-order



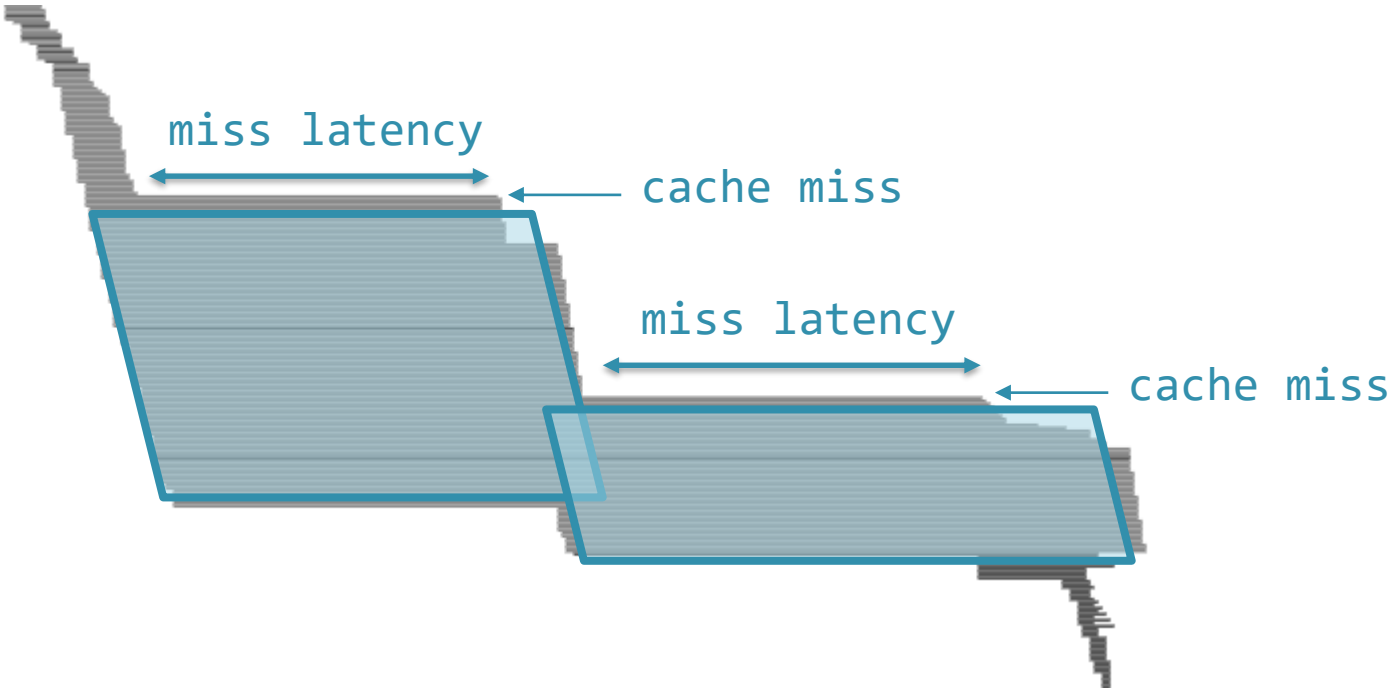
Example: Branch Misprediction

- Flushed instructions are shown as dark ones



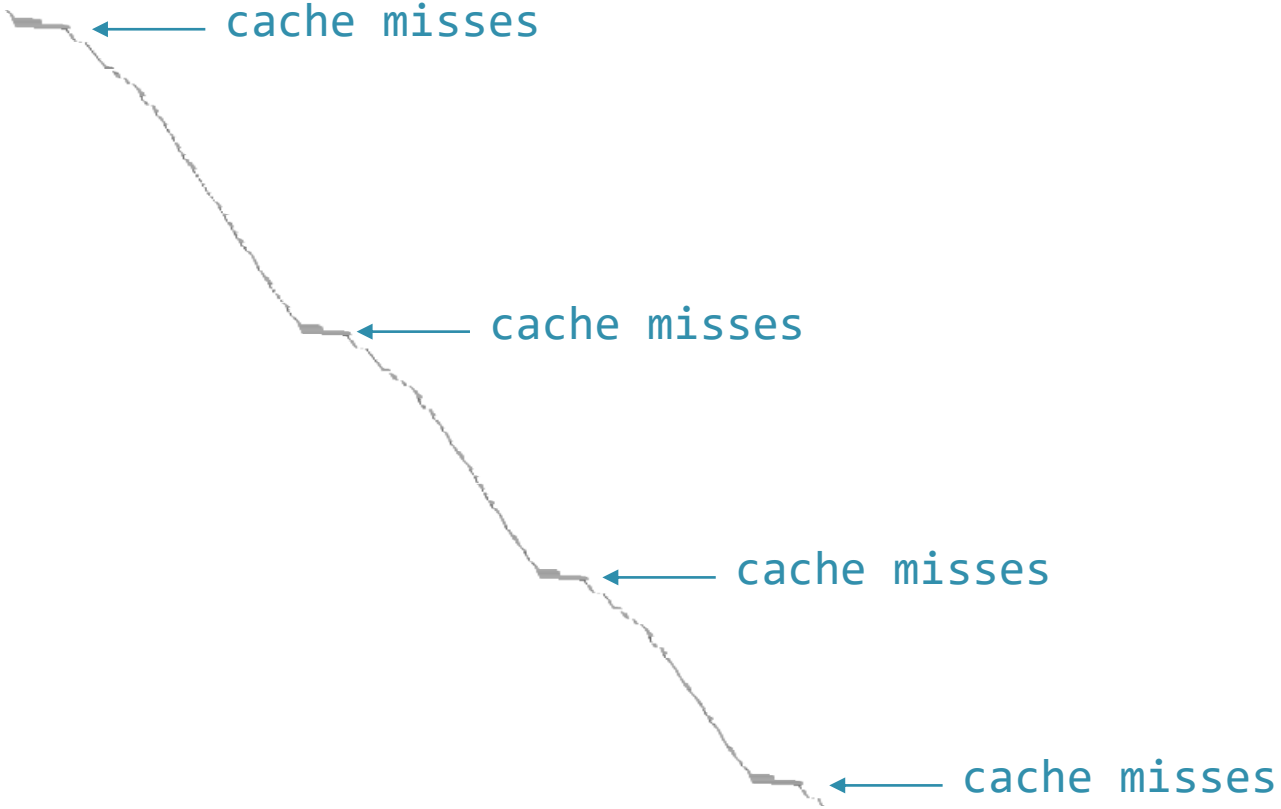
Example: Cache Misses

- A cache miss is typically shown as a diamond-like shape when the image is zoomed out as follows



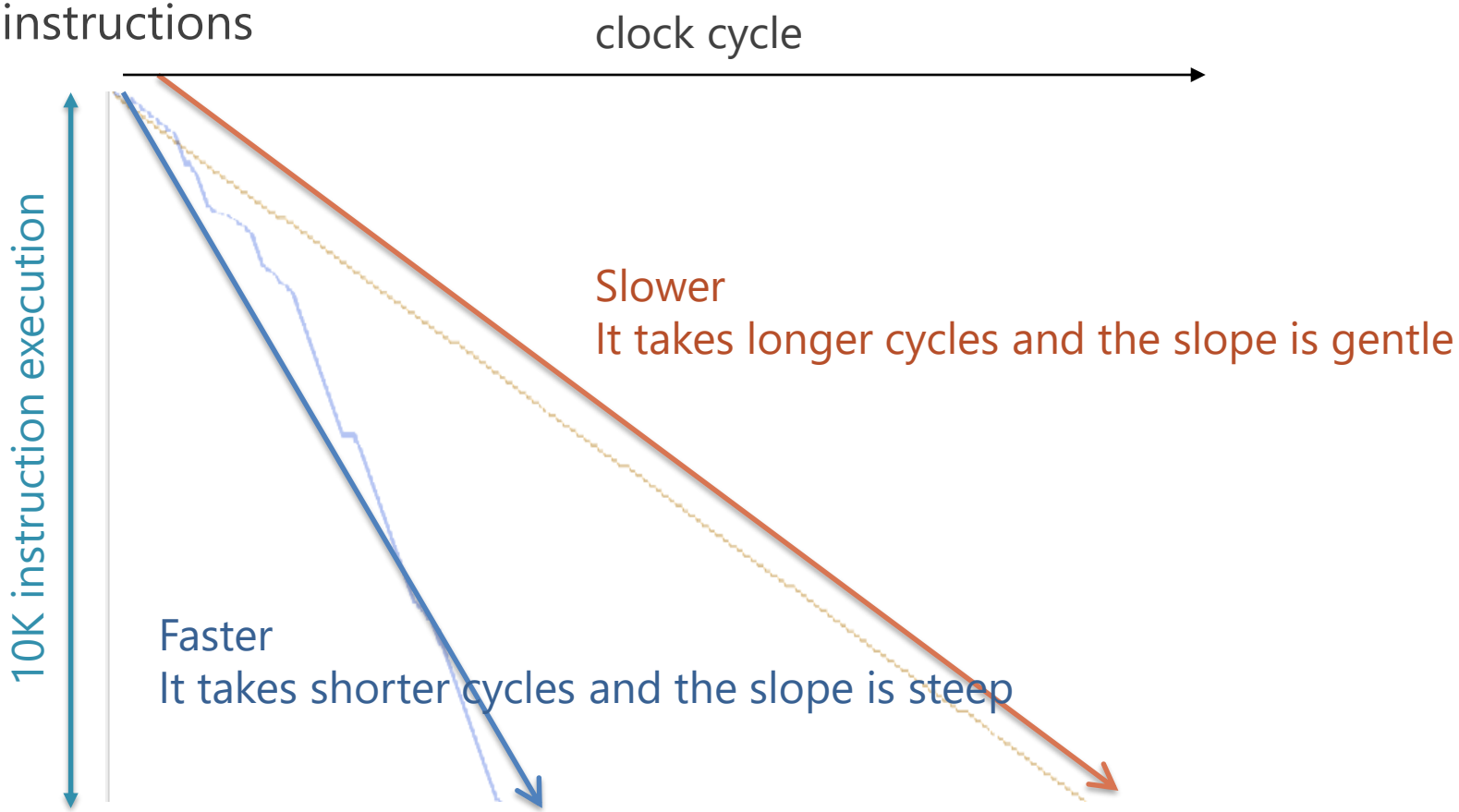
Example: Cache Misses

- As it is zoomed out more, the pipeline is typically shown as follows
  - This is the pipeline behavior of MCF in SPECCPU 2006
  - This figure shows the performance is degraded by the cache misses



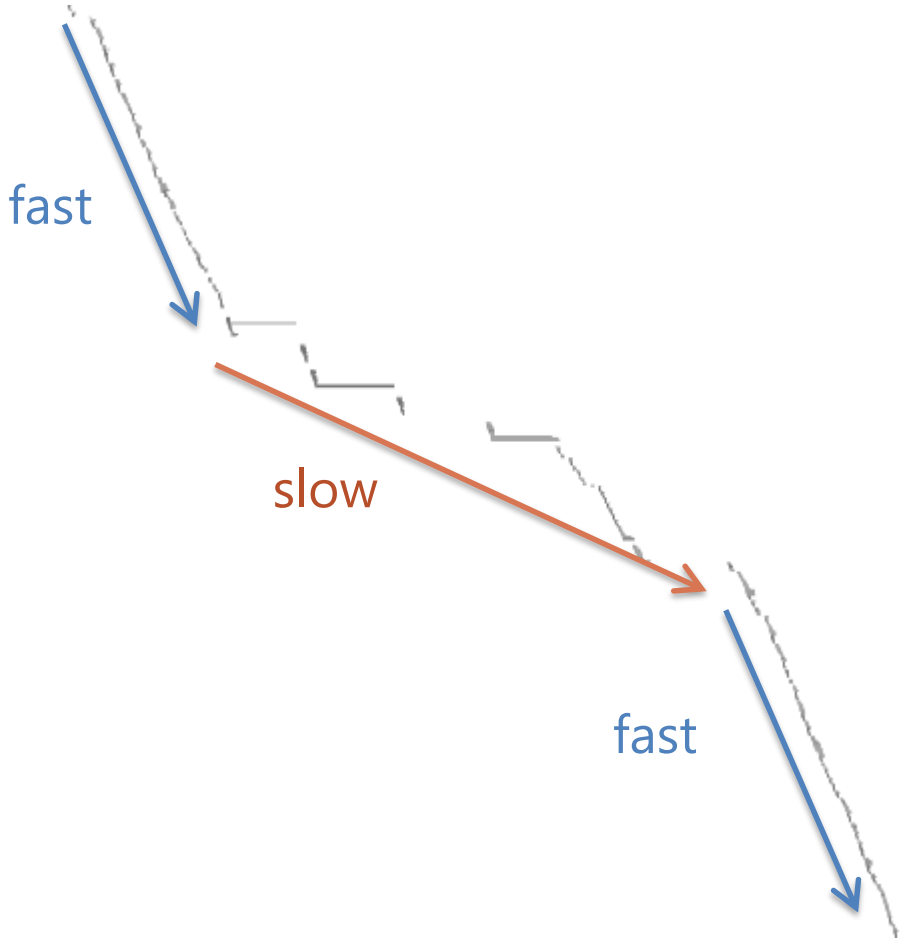
Example: Execution Speed

- The slope of a pipeline shape roughly represents the execution speed (IPC).
- The following two pipelines show the execution of the same 10K instructions



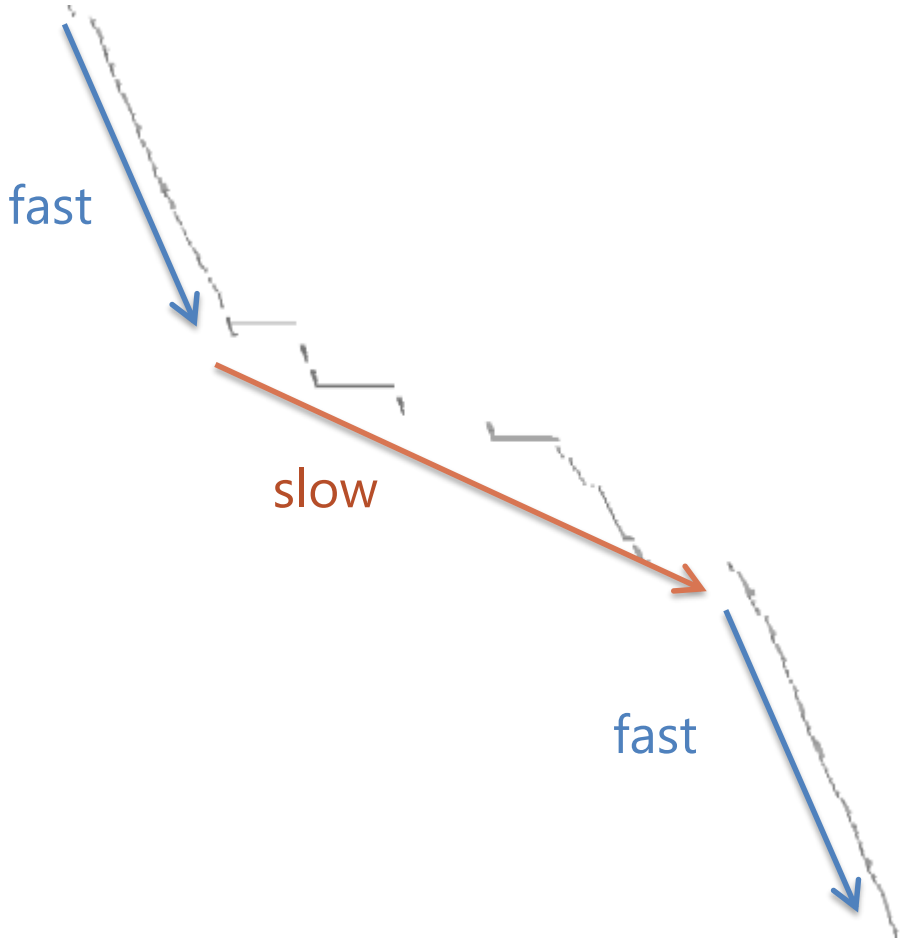
The slope of a pipeline shape roughly represents the execution speed (IPC)

- You can see the transition in the execution speed for each part of the program as follows



The slope of a pipeline shape roughly represents the execution speed (IPC)

- It is not accurate because flushed instructions are also shown.
- If you want to compare accurately, use "Hide flushed ops" option from the right click menu



Outline

1. A brief explanation of how to use
2. Typical visualization examples
3. Use cases
    1. Grasping the pipeline behavior
    2. Comparing pipelines

Grasping the pipeline behavior

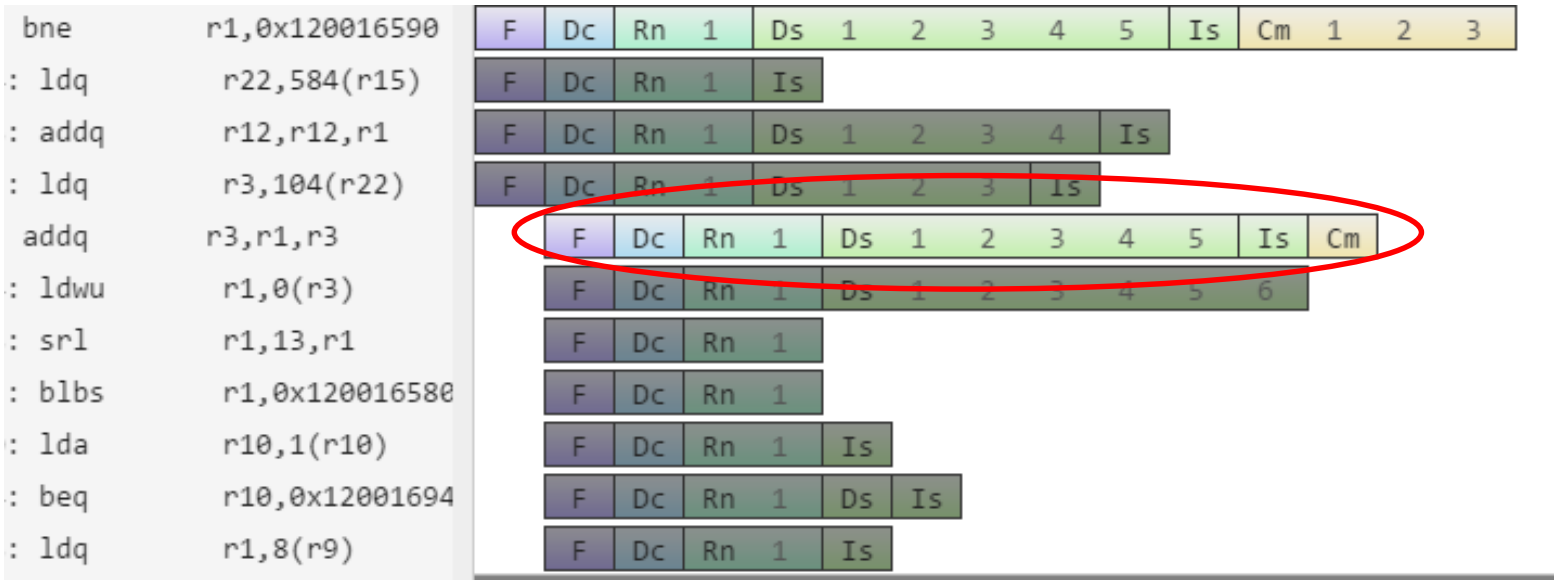
- The pipeline visualization makes it easy to grasp the pipeline behavior
  - Explain this by some use cases
- Let's suppose you newly add speculative execution with branch prediction
  - (Of course, gem5 already has this feature
  - Something wrong happens in recovery from mispredictions

Investigating with a log

```
O3PipeView:fetch:25069250:0x120016358:0:177:addq        r12,r12,r1
O3PipeView:decode:25069500
O3PipeView:rename:25069750
O3PipeView:dispatch:25070250
O3PipeView:issue:25071500
O3PipeView:complete:25071750
O3PipeView:retire:0:store:0
O3PipeView:fetch:25069250:0x12001635c:0:178:ldq         r3,104(r22)
O3PipeView:decode:25069500
O3PipeView:rename:25069750
O3PipeView:dispatch:25070250
O3PipeView:issue:25071250
O3PipeView:complete:25071500
O3PipeView:retire:0:store:0
O3PipeView:fetch:25069500:0x120016360:0:179:addq        r3,r1,r3
O3PipeView:decode:25069750
O3PipeView:rename:25070000
O3PipeView:dispatch:25070500
O3PipeView:issue:25072000
O3PipeView:complete:25072250
O3PipeView:retire:25072500:store:0
O3PipeView:fetch:25069500:0x120016364:0:180:ldwu        r1,0(r3)
O3PipeView:decode:25069750
O3PipeView:rename:25070000
O3PipeView:dispatch:25070500
O3PipeView:issue:25072250
O3PipeView:complete:0
O3PipeView:retire:0:store:0
O3PipeView:fetch:25069500:0x120016368:0:181:srl         r1,13,r1
O3PipeView:decode:25069750
O3PipeView:rename:25070000
O3PipeView:dispatch:25070500
O3PipeView:issue:0
O3PipeView:complete:0
O3PipeView:retire:0:store:0
```

- For investigating your implementation, you probably:
  - Check custom logs or your "printf" outputs such as the left example
  - It records when/what instructions are flushed.
- It's very difficult to detect which point is incorrect from such text logs.

Investigating with visualization



- By visualizing it, you can easily notice the incorrect point.
  - ◇ There is the light instruction (not flushed) between the dark flushed instructions.
- Although this is an artificial example,
  - ◇ visualization gives us a lot of hints intuitively

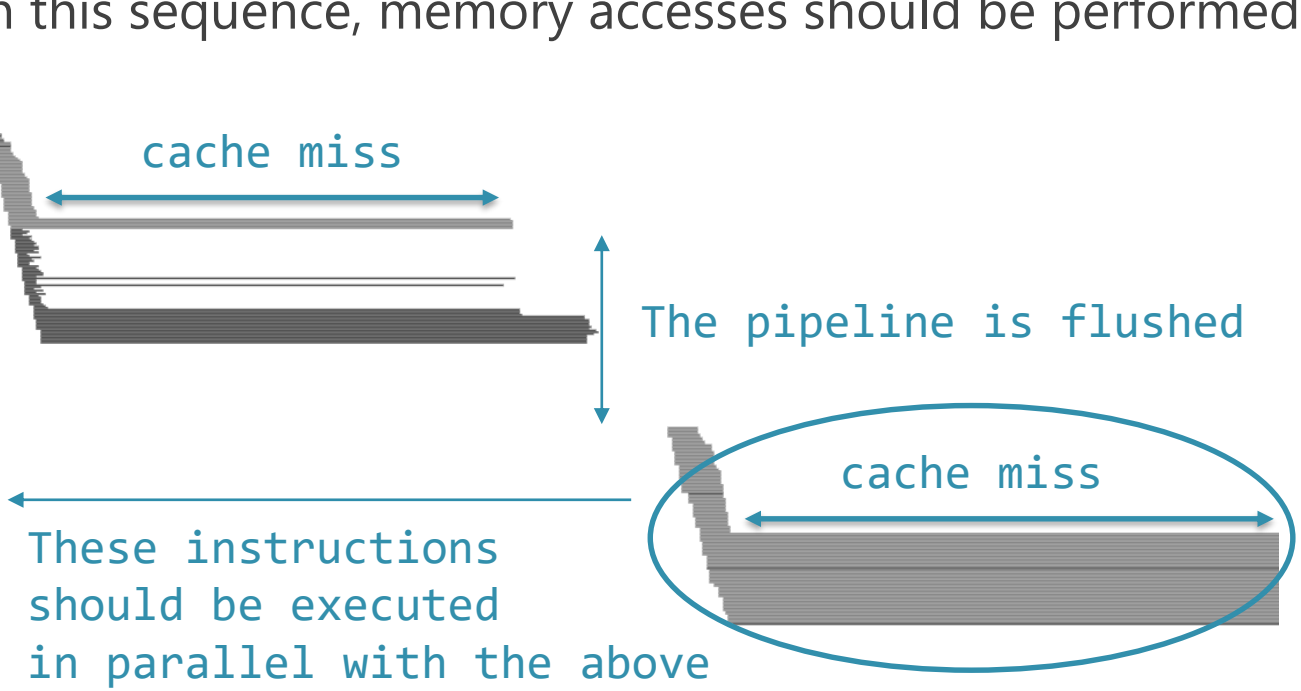
Another example: memory level parallelism

- One of my friends researched a theme related to memory level parallelism
  - In short, his method improves the performance by performing multiple memory accesses in parallel
- He enlarged the size of the OoO scheduling window so that more memory accesses are performed in parallel
  - But, the performance is not improved

Another example: memory level parallelism

- He realized that something wrong happened from the following zoomed out image,
  - because the shape is unnatural

- He realized that the pipeline was flushed on a cache miss
  - In this sequence, memory accesses should be performed in parallel



The cause of the flush

- He examined the flushed instruction in detail and found the cause.
- This was because he used Alpha ISA
  - In Alpha architecture, TLB miss causes a trap and the pipeline is flushed
  - On a cache miss, a TLB miss often occurs
  - So memory accesses cannot be performed in parallel
- It is not easily noticed simply by observing the counters in gem5.
  - The shape or pattern of visualized pipelines often tell us hints.

Outline

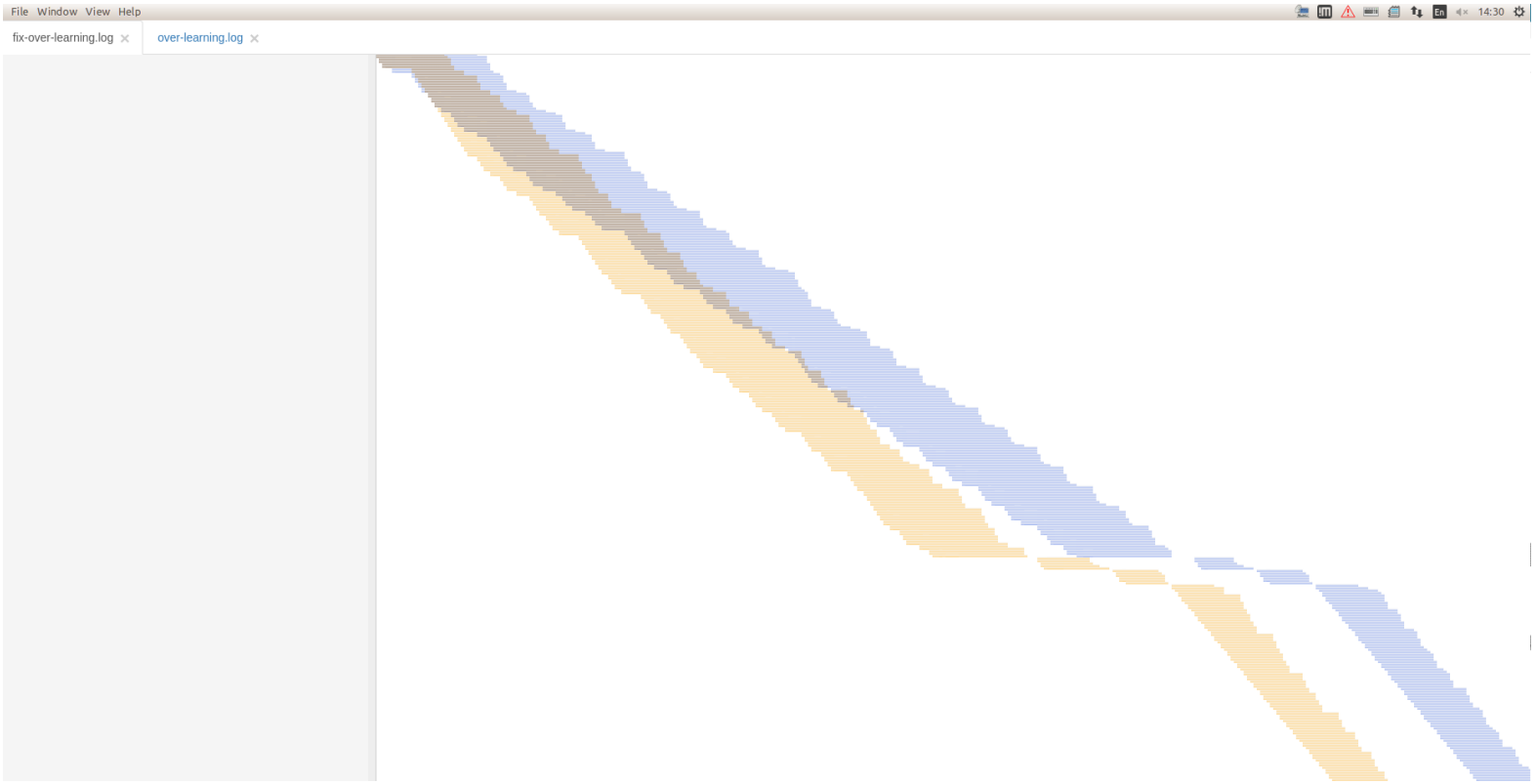
1. A brief explanation of how to use
2. Typical visualization examples
3. Use cases
    1. Grasping the pipeline behavior
    2. **Comparing pipelines**

Comparing pipelines

- Let's suppose
  - your new method seems to work correctly,
  - but it does not improve the performance as you expected.
- Konata can compare two pipelines.
  - It is useful when investigating the above situation.

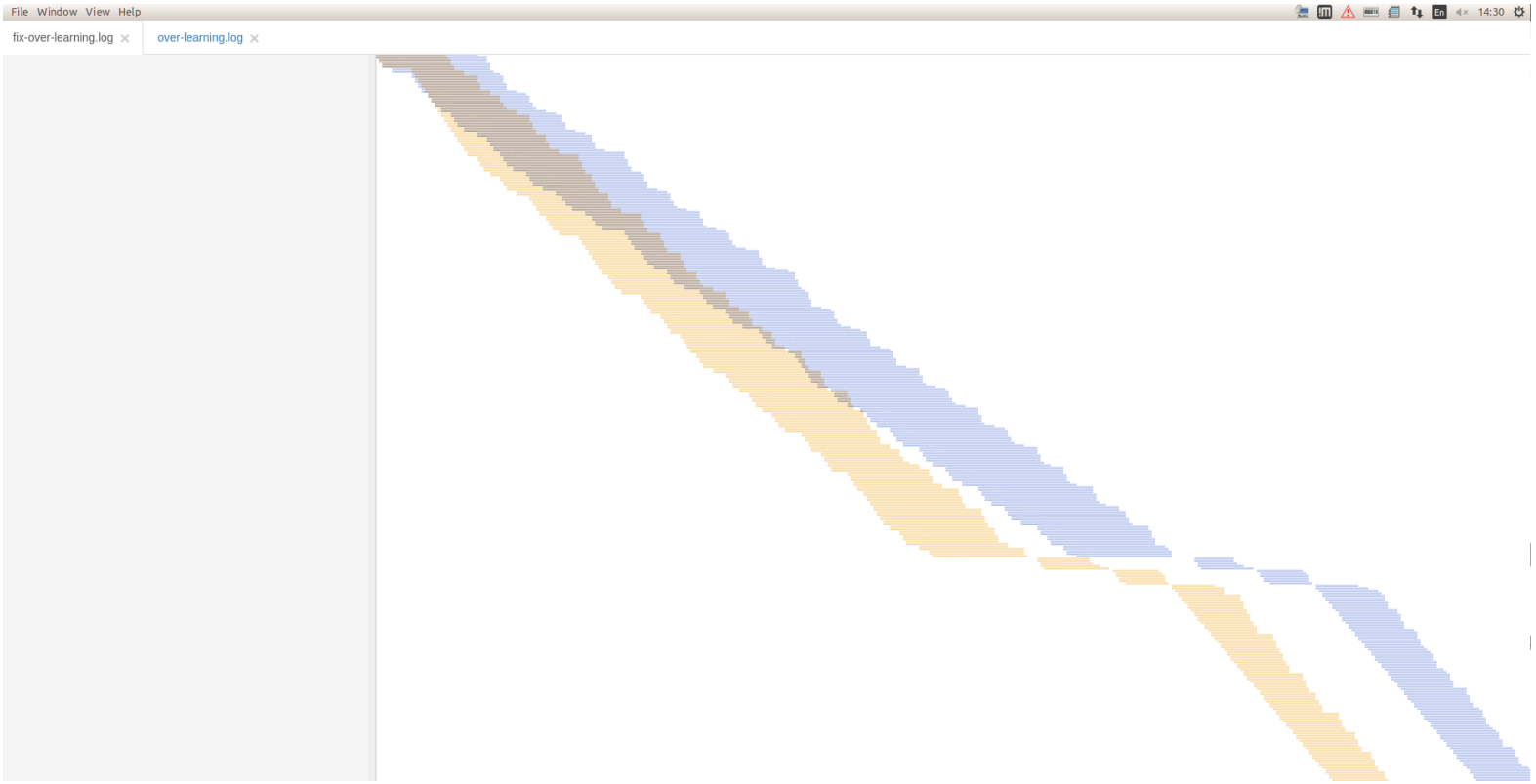
Example of comparing

- My friend implemented the new method to the baseline CPU.
- Konata can show two pipelines overlapping.
  - Blue shows a baseline pipeline
  - Orange shows a pipeline with a new method



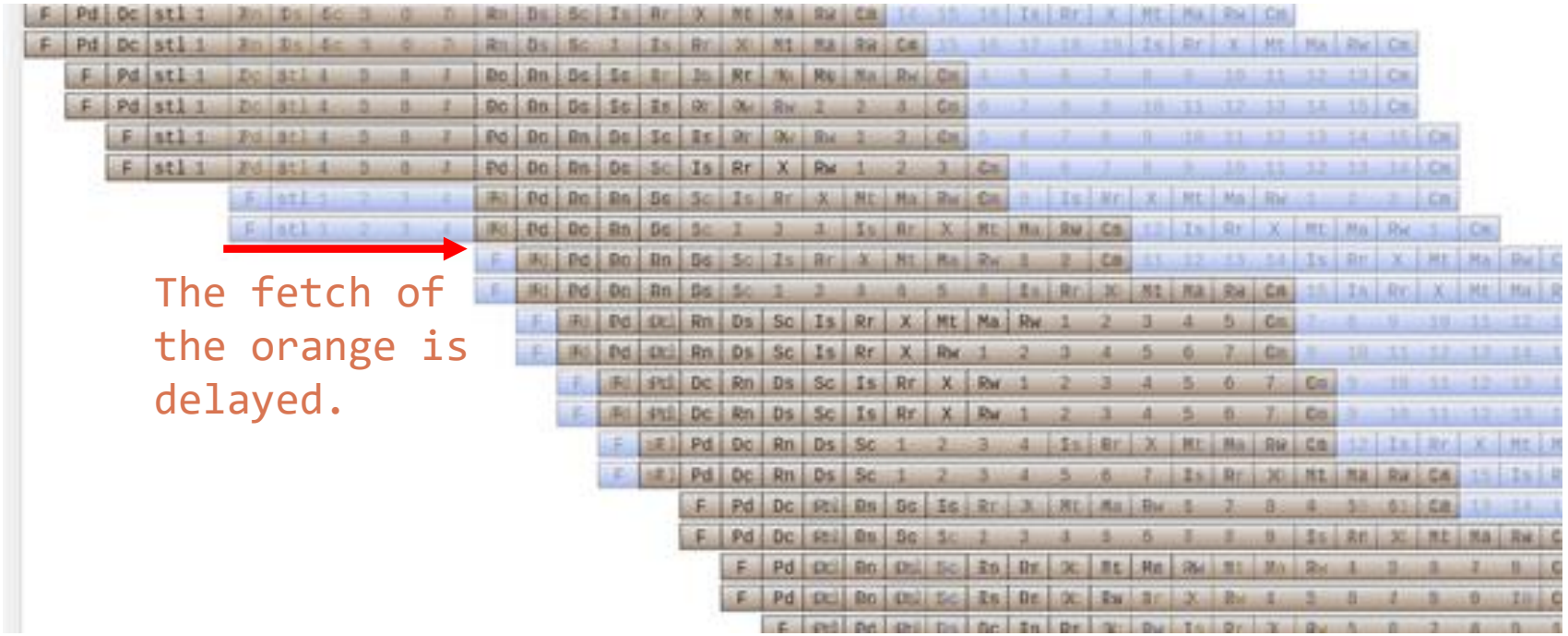
Example of comparing

- The orange one (new) is basically faster than the blue one (baseline)



Example of comparing

- In the zoomed-in image,
  - in some places, the fetching of the orange is unreasonably delayed.
  - This was caused by a bug in fact.



Comparing pipelines

- Visual comparison is very effective for analysis when adding new features to gem5.
  - If the performance is not improved as expected, something is delayed.
  - You can detect such parts by visual comparison.
    - It is easy to see which part is different.

Conclusion

- It is generally difficult to investigate the cause of a bug related to the performance.
  - Especially, when you have no idea what happened.
- In such cases, visualization is very useful.
  - This presentation introduced the pipeline visualization in gem5
- Please try it!
  - It is simply fun to see how the processor works.
  - https://github.com/shioyadan/Konata/releases