Towards Systematic Low-Overhead Tracing: Control-Flow-Sampling (CFS) Guided Tracing

Sampling vs. Tracing

```
int a(){
  //some computation
  return c(x);
}
int b(){
  //some computation
  return c(x)+c(y)+c(z);
}
int c(arg){
  //some computation
  return result;
}
int main(){
   return a() + b();
}
```

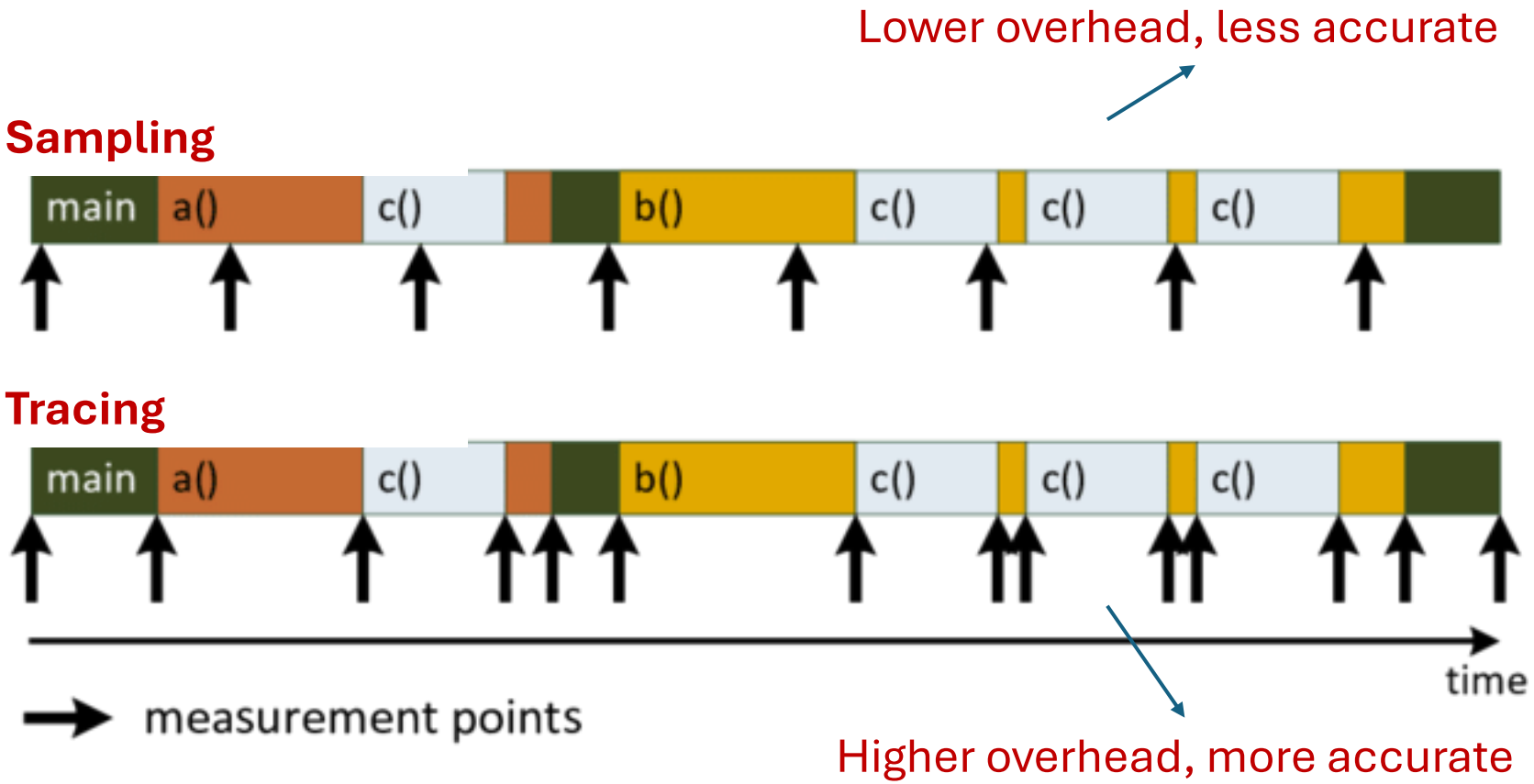


Figure adapted from: Molka, D. (2017). *Performance analysis of complex shared memory systems*

Use low-overhead sampling to guide high-precision tracing?

I think I need some help

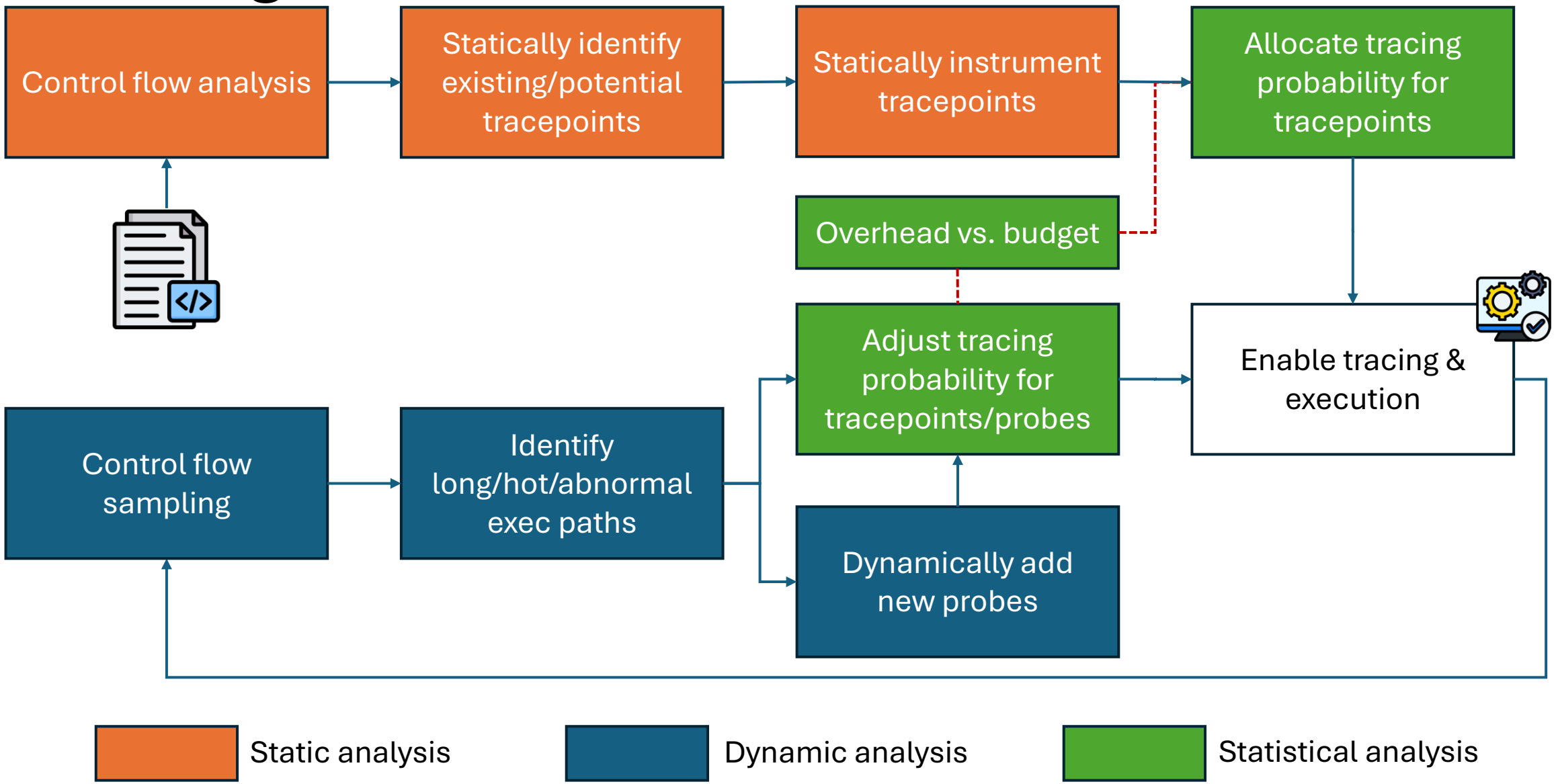
Sampling (like police on patrol)

Tracing (like police in action)

Control-Flow-Sampling (CFS) Guided Tracing: Intuitions

- Start with **statically** identifying existing/potential tracepoints (e.g., methods or basic blocks)
- Use tracing **probability** to control overhead budget: statistical tracing
- Use low-overhead **control flow sampling** to identify **worthy-to-trace** and **costly-to-trace** program units: adjusting tracing probability
- Use high-precision **tracing** to collect runtime data for important units
- Use overhead budget and **overhead monitoring/estimation** to control tracing probability

Control-Flow-Sampling (CFS) Guided Tracing



Statically determine where to trace (uncertainty points)

- (High-level) function entry/exit (arguments, returns)
- Basic code blocks (selectively)
- Branch points (conditional statements)
- Loop iterations (performance bottlenecks)
- Error handling (e.g., try/catch blocks)
- Resource allocation (acquiring/releasing critical resources)
- API/RPC returns

To be determined by representative use cases from industry: to discuss

Control flow sampling (hardware or software based)

- **Last branch records (LBRs)**
  - Recording the **last 8-32 branches** in model-specific registers (MSRs).
  - Nearly zero overhead for recording branches in MSRs.
  - We can **sample the MSRs** to obtain control flow (branches) info.
    - Sampling frequency determines overhead.
  - Supported by **Intel, AMD, and ARM64**
- **IntelPT or PTWrite Snapshots**
  - **Taken or Not-Taken (TNT)** of branches; target address of indirect branches.
  - Default IntelPT traces all branches: too much data.
    - Good for post-mortem analysis but not for on-the-fly analysis.
  - The **snapshot** option: uses a small buffer to store a snapshot
  - Supported by **Intel**.
- **Call stack sampling**
  - Sampling at the call stack level (less precise)
  - No special hardware support needed

To be determined by representative use cases from industry: to discuss

Dynamically identify interesting exec paths to trace

- **Long-running paths**
  - Increase tracing probability
  - Add new probes
- **Frequently-executed paths**
  - Decrease tracing probability (to reduce overhead)
  - Increase tracing probability of caller (to find out why frequent)
- **Abnormal/rare/ unstable (perf-varying) paths**
  - Increase tracing probability
  - Increase tracing probability of caller and callees

To be determined by representative use cases from industry: to discuss