# Motivation

- Examples Bounded Loops

```
for (i = 0; i < max; i++) { do work }
while (i > blah) { … };
do { work } while {i}
```

- Guidelines:
  - Lots of academic work on complex loops
    - polynomial invariants, Grobner basis and more ← fun but lets stick to basic ax+c for now.

- Agenda:
  Review terms, goals, etc.
  Approach #1 (by the books)
  Approach #2 (compiler aided)
  Approach #3 (instruction based)
  Discuss

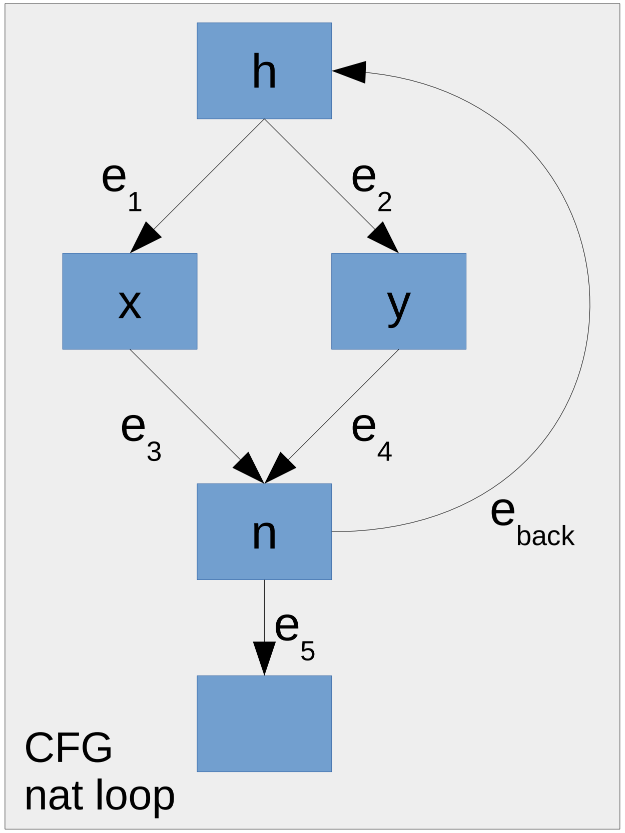
i and j are Induction variables

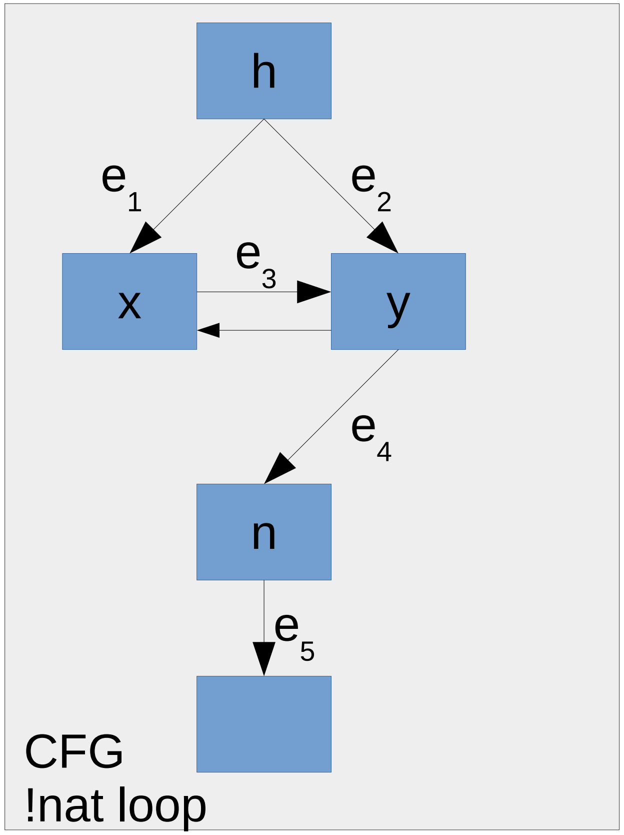
```
int array[10] = init
int max = 10, foo = blah, bar = blah;

for (i = 0; i < max; i++) {
     int j = i * foo + bar;

     value = bpf_map_lookup_elem(&map,
&key);
     if (value > 0)
           sum += array[j]
     else
            sum -= array[j]
}
```

h is a header node
$e_{back}$ backedge n->h

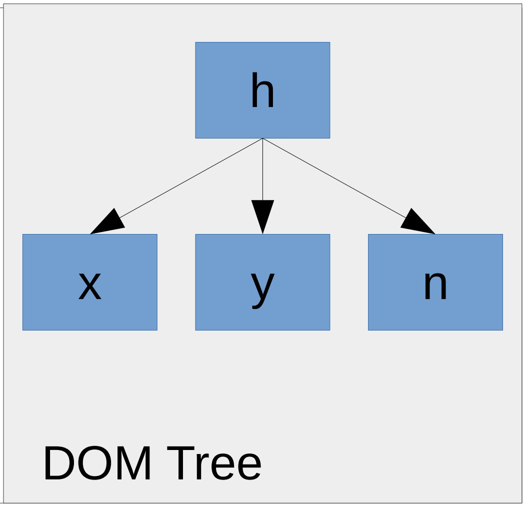




h dominates n, x, y
Natural loop: the set of nodes x, where h dom x with a path from x to n _not_ containing h.

*intuition: Does not have multiple goto's into loop.*

Find Natural Loop Algorithm:
1. Compute CFG and Dominator Tree
2. Find back edges
3. Find the natural loop using DOM Tree



# Approach #1: by the book

https://git.kernel.org/pub/scm/linux/kernel/git/bpf/bpf-next.git/ wip/bpf-loop-detection

- Build CFG
- Build DOM Tree
- Detect and abort on irreducible loops
- Find loops (back edges)
- For Each Loop
  - Find induction variables (pattern matching)
  - Verify bounds on loop induction variable terminate
  - “run” loop with worst case bounds, pruning works, array index worst case.

```
hdr:
    <do stuff>
    if (i != x) goto hdr
```

```
hdr:

    <do stuff>
    if (i != x) goto out
    <do more stuff>
    goto hdr out:
    <outside loop>
```

Challenge: Many LLVM loop patterns. At the moment we do pattern matching and can extend these but fragile.

PROP1: General forest of Induction variables or SCEV needed.

# Approach #2: Compiler Aided

https://git.kernel.org/pub/scm/linux/kernel/git/bpf/bpf-next.git/ wip/bpf-loop-detection

- Limit types of loops constructed by LLVM

```
hdr:
    <do stuff>
    if (i != x) goto hdr
```

- Easy to pattern match if LLVM plays along
- Still need to do full verification of natural loops (build DOM tree, etc.) and find induction variables. But somewhat easier because of friendly LLVM.

  -

# Approach #3: New instructions

- Loop specific instructions
  - Denote loop blocks with instructions loop/end
  - Requires LLVM backend to convert unstructured gotos into structured loops. DOM tree no longer required replaced with strict hierarchy of blocks.
  - Ensure goto’s into loop blocks fail, overlapping blocks not allowed, induction variable tracking still required.

BPF instruction label, NOP in JIT

BPF *JLP* instructions, jumps to scoped paired BPF_JLOOP_LABEL. Verifier will need to track pairs and replace with proper jumps after verification.

```
► BPF_JMP_LOOP(BPF_JLOOP_LABEL)

  […] ← (jumps into block not allowed)

► BPF_JMP_LOOP(BPF_JLPEQ, BPF_REG_0, 0)
```

## Discuss

Decide how to proceed and get loop support.