Dynamic and Adaptive Calling Context Encoding

Jianjun Li, Zhenjiang Wang, Chenggang Wu
State Key Laboratory of Computer Architecture
Institute of Computing Technology, CAS

Wei-Chung Hsu
Department of Computer Sciences,
National Taiwan University

Di Xu
IBM Research - China

CGO 2014, Orlando, Florida

Introduction

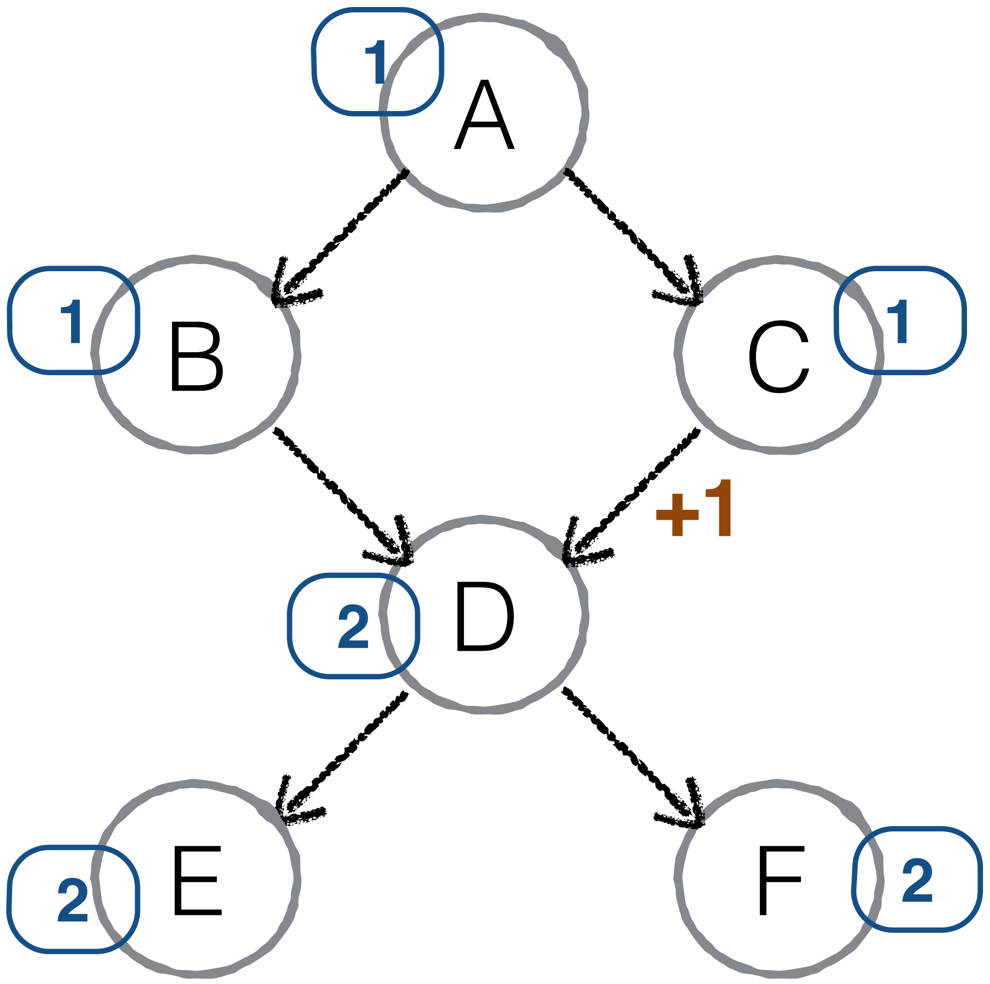
- Calling contexts are the sequence of active functions on call stack
- Calling contexts play an important role in a wide range of software development processes.
  - Testing
  - Debugging and error reporting
  - Program analysis
  - Security enforcement

Existing Approaches

- Accurate calling context
  - Stack Walking, Calling context trees *or* calling context up trees
    - High overhead
  - Precise calling context encoding (ICSE'2010)
    - Static encoding method, work only on complete call graph
    - Unable to handle dynamic loading and virtual dispatch
- Inaccurate calling context
  - Inferred Call Path Profiling (OOPSLA '09)
    - Low overhead but not precise enough
  - Hash based path encoding: Probabilistic Calling Context (OOPSLA '07), Breadcrumbs (PLDI'2010)
    - Trade accuracy to performance

Background: Calling Context Encoding

- Calling context encoding
    - Based on Ball-Larus path encoding algorithm (BL algorithm)
    - Encode a call path to an integer
    - Accurate calling context
    - Low overhead



Background: Calling Context Encoding

- **Problems:**
  - Static encoding method, work only on complete call graph
  - Unable to handle dynamic loading and virtual dispatch
  - Need profiling runs or pointer analysis to identify the targets of indirect calls
  - Not efficient in encoding space

Outline

- Our Goals and Key Challenges
- Dynamic Encoding Method
- Adaptive Encoding Method
- Experimental Results
- Summary

Our goals

A dynamic and adaptive context encoding algorithm:

☐ Does not need extra profiling runs or static program analysis

☐ Handle dynamic loadings

☐ Adaptive to program behavior changes

☐ Efficient in encoding space and time

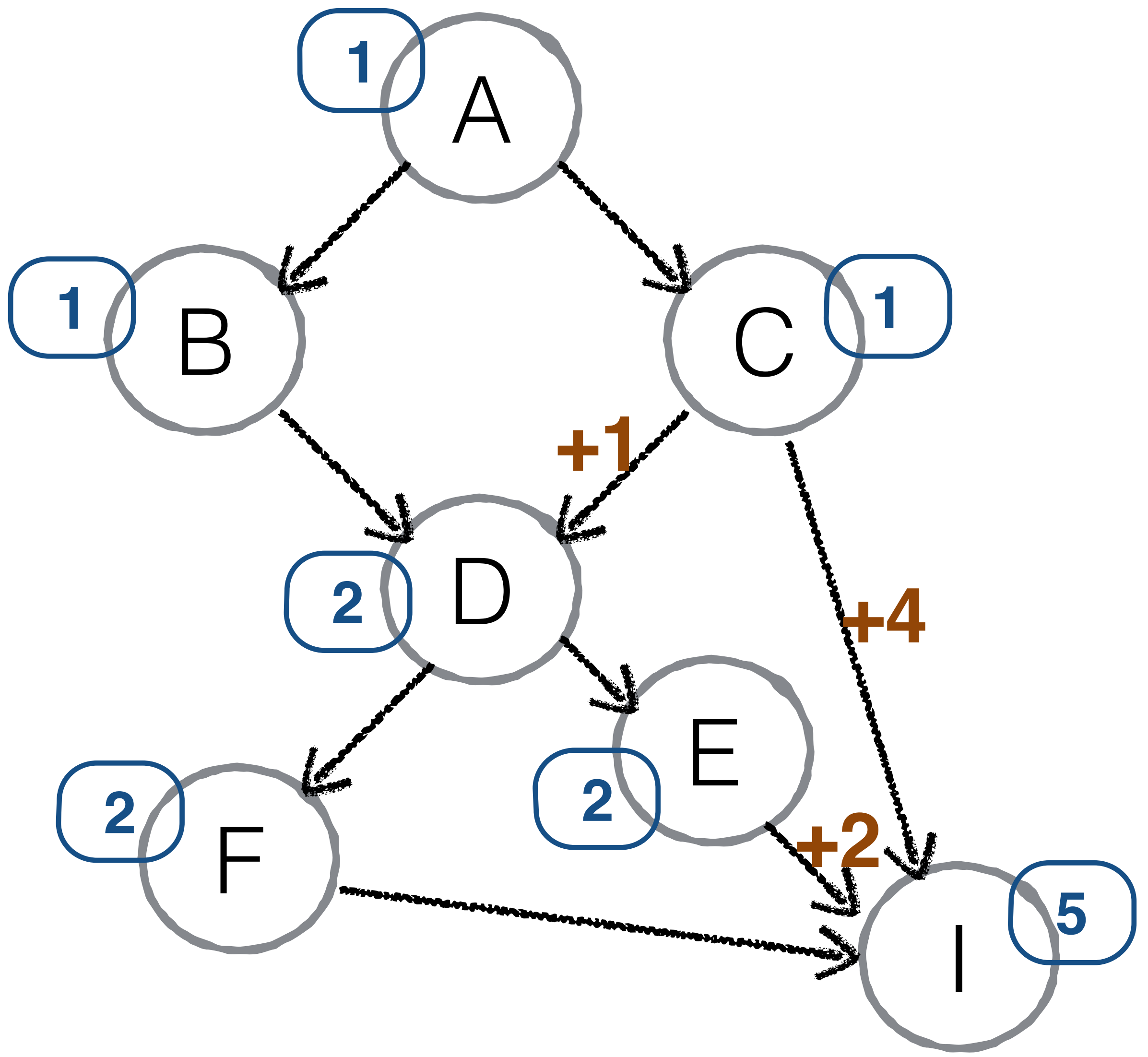
☐ Accurate context information

Key Challenges

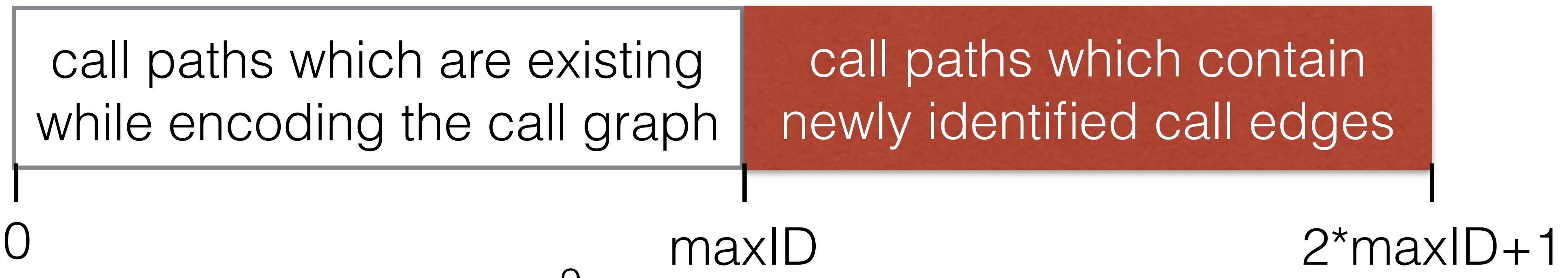
- How to handle newly identified call edges?
  - Indirect call paths
  - Dynamic loadings
- How to ensure the collected path *id*s be correctly decoded?
  - The encodings of call edges may change after adaptive encoding.

Dynamic Encoding Method Overview

Call Graph:
maxID=4

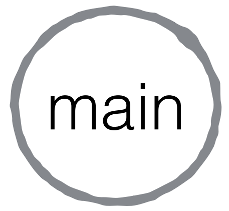


Encoding Space



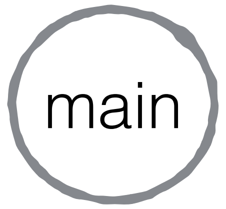
call paths which are existing while encoding the call graph
call paths which contain newly identified call edges
0
maxID
2*maxID+1

Dynamic Encoding



Initially, the call graph only contain the entry function “main”.

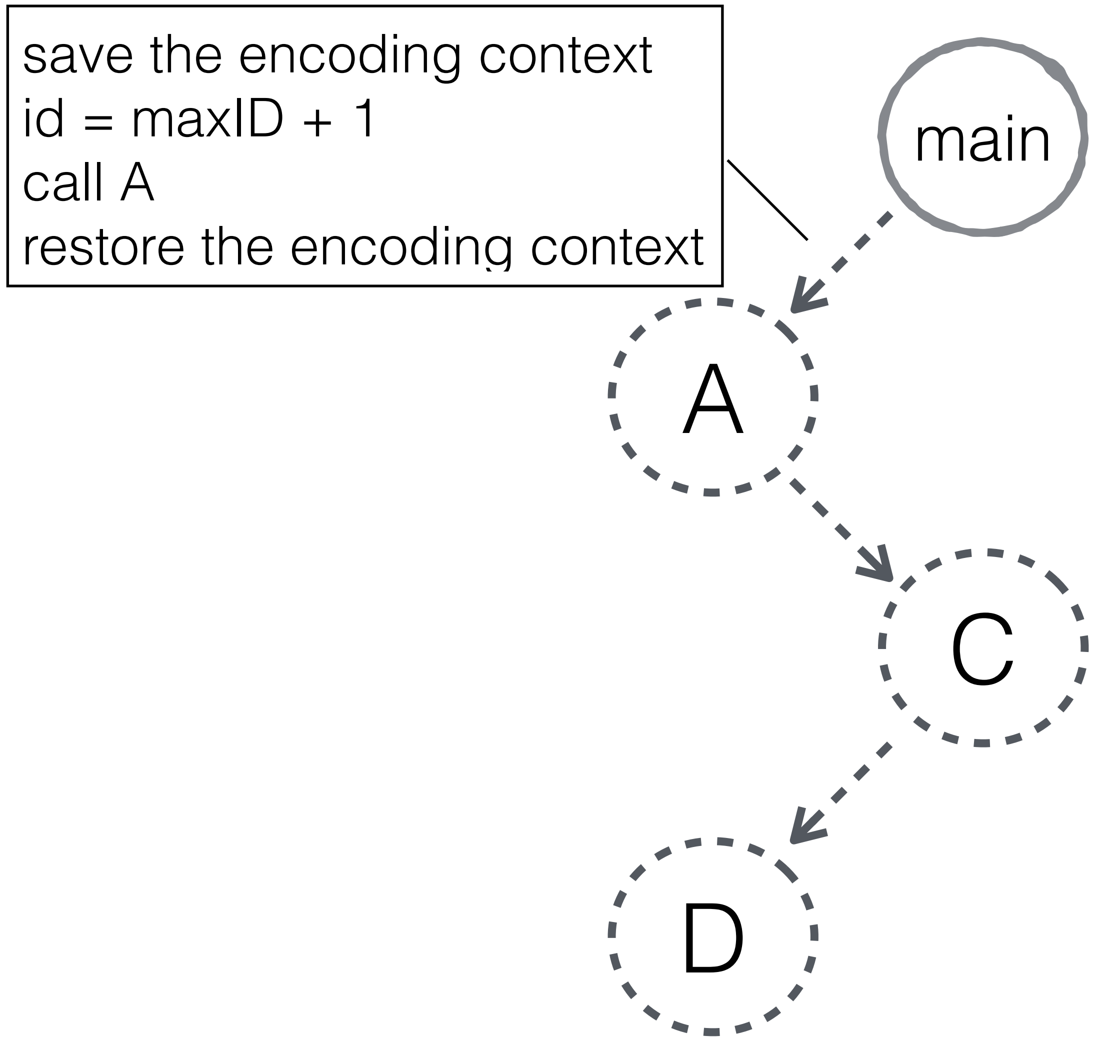
Dynamic Encoding



Initially, the call graph only contain the entry function “main”.

Replace all function call instructions with “call rtHandler”.

Dynamic Encoding



Initially, the call graph only contain the entry function “main”.

Replace all function call instructions with “call rtHandler”.

In rtHandler, update the call graph and instrument that edge.

Adaptive Encoding

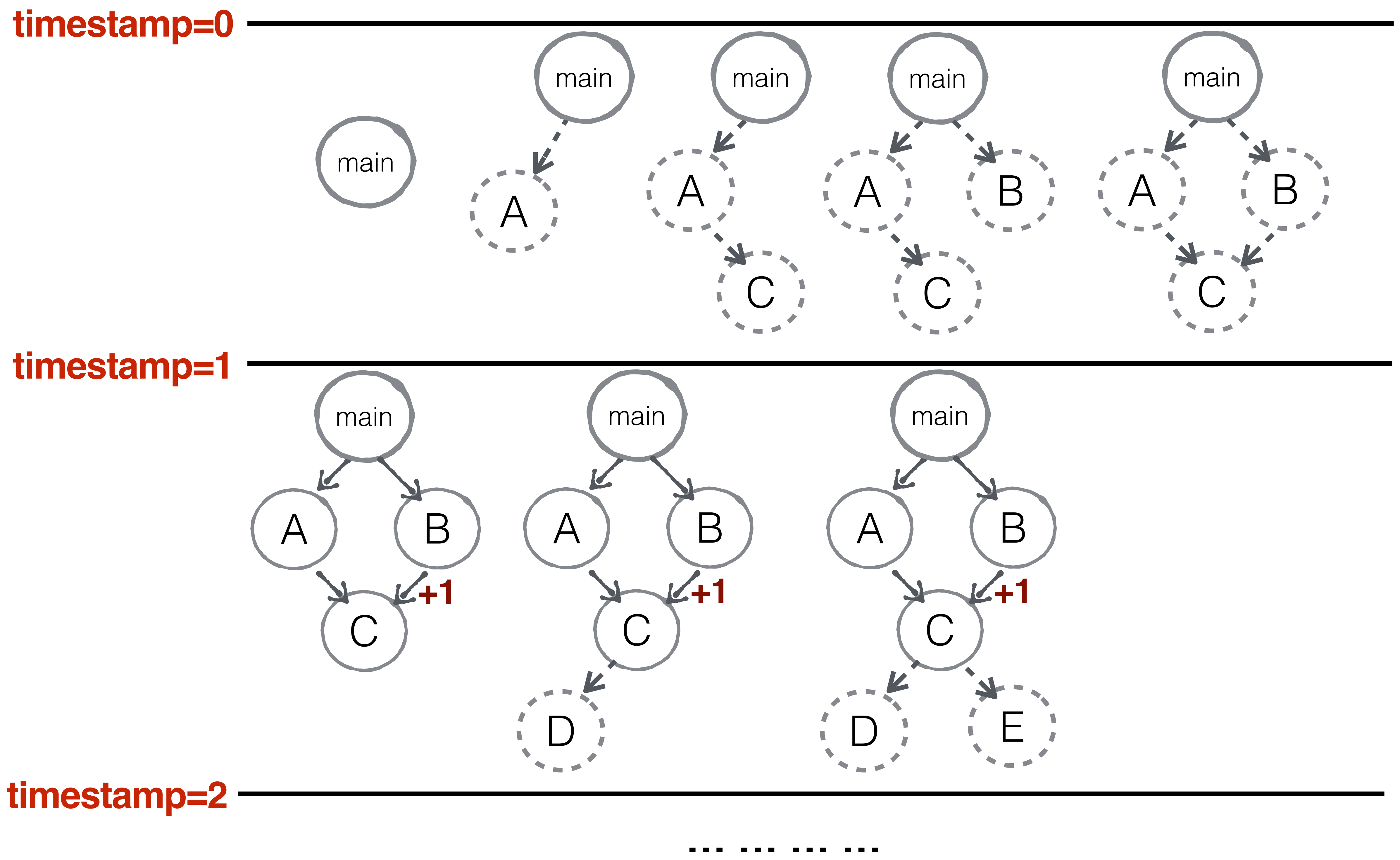
- Why adaptive encoding?
  - reduce the runtime overhead
  - adaptive to program's runtime behavior
- Trigger conditions of adaptive encoding:
  - The number of identified call edges reaches a threshold.
  - The frequently invoked call paths have changed.
  - The helper stack is frequently accessed.

Adaptive Encoding

- Adaptive encoding process:
  - Decode and analyze the collected contexts, mark the frequently invoked call edges.
  - Encode the call graph, and adjust the encodings according to the invocation frequency.
  - Instrument the program with the new encodings.

Adaptive Encoding

Adaptive Encoding

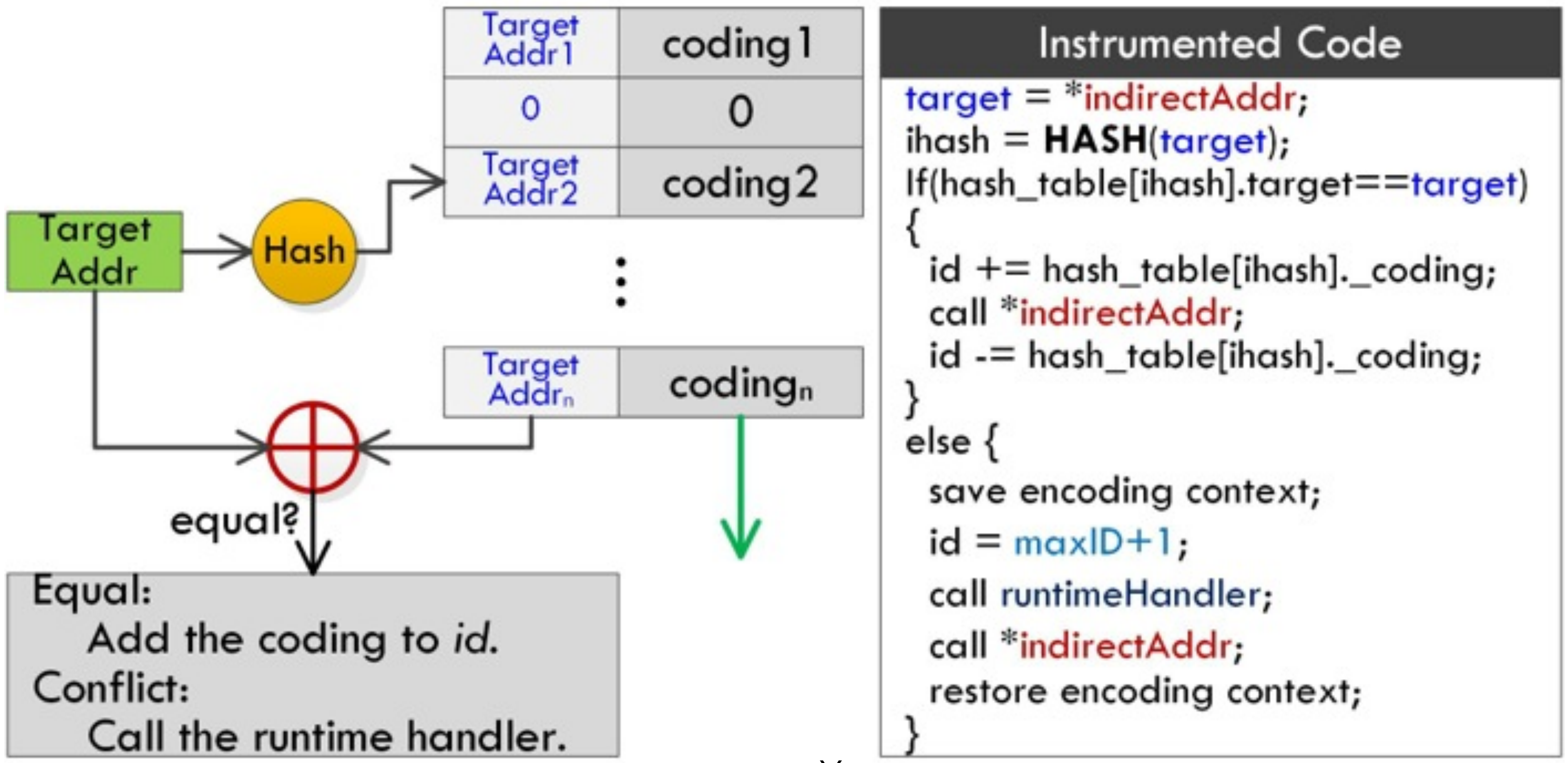


Recursive Calls

- BL path encoding algorithm only woks on acyclic graph.
- Recursive call paths will be encoded into range [maxID+1, 2*maxID+1].
- For highly repetitive recursive calls, the saved encoding contexts will be compressed.

Indirect Calls

- An indirect call may have multiple targets.
- After re-encoding, the identified targets are instrumented separately.

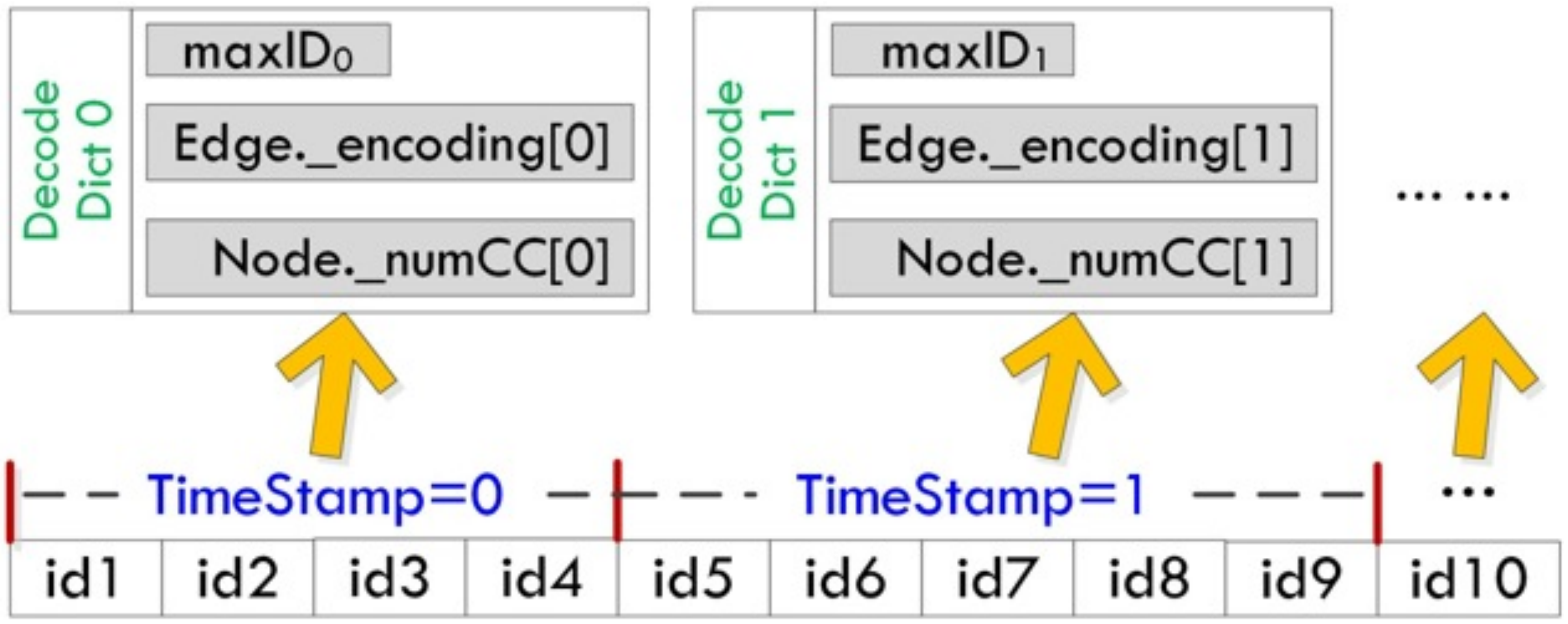


```
target = *indirectAddr;
ihash = HASH(target);
If(hash_table[ihash].target==target)
{
  id += hash_table[ihash]._coding;
  call *indirectAddr;
  id -= hash_table[ihash]._coding;
}
else {
  save encoding context;
  id = maxID+1;
  call runtimeHandler;
  call *indirectAddr;
  restore encoding context;
}
```

X

Decoding Mechanism

- Call graph is growing dynamically as the program runs.
- To correctly decode the recorded context, we need the exact call graph and encoding information when the context is recorded.



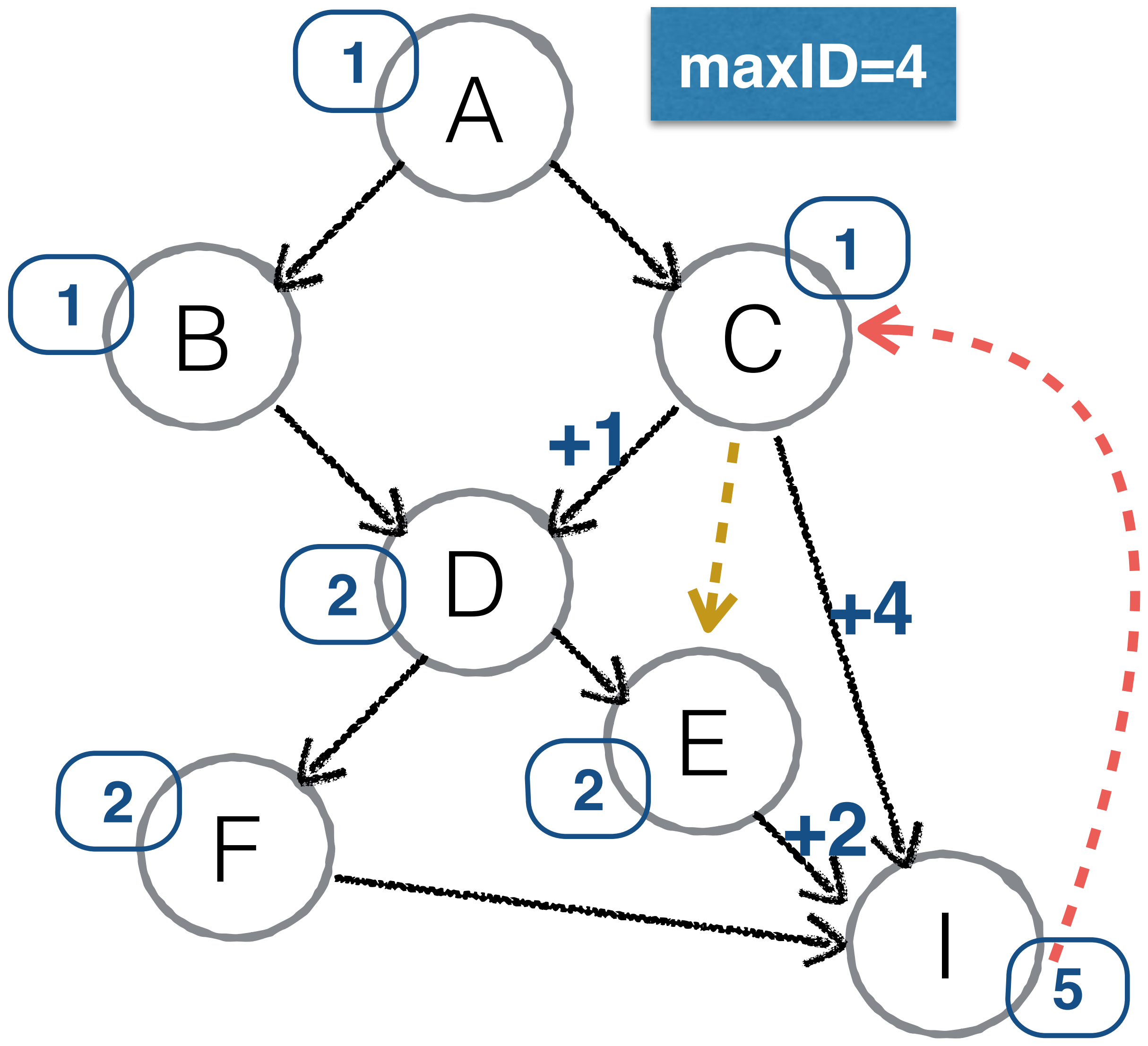
Decoding Algorithm

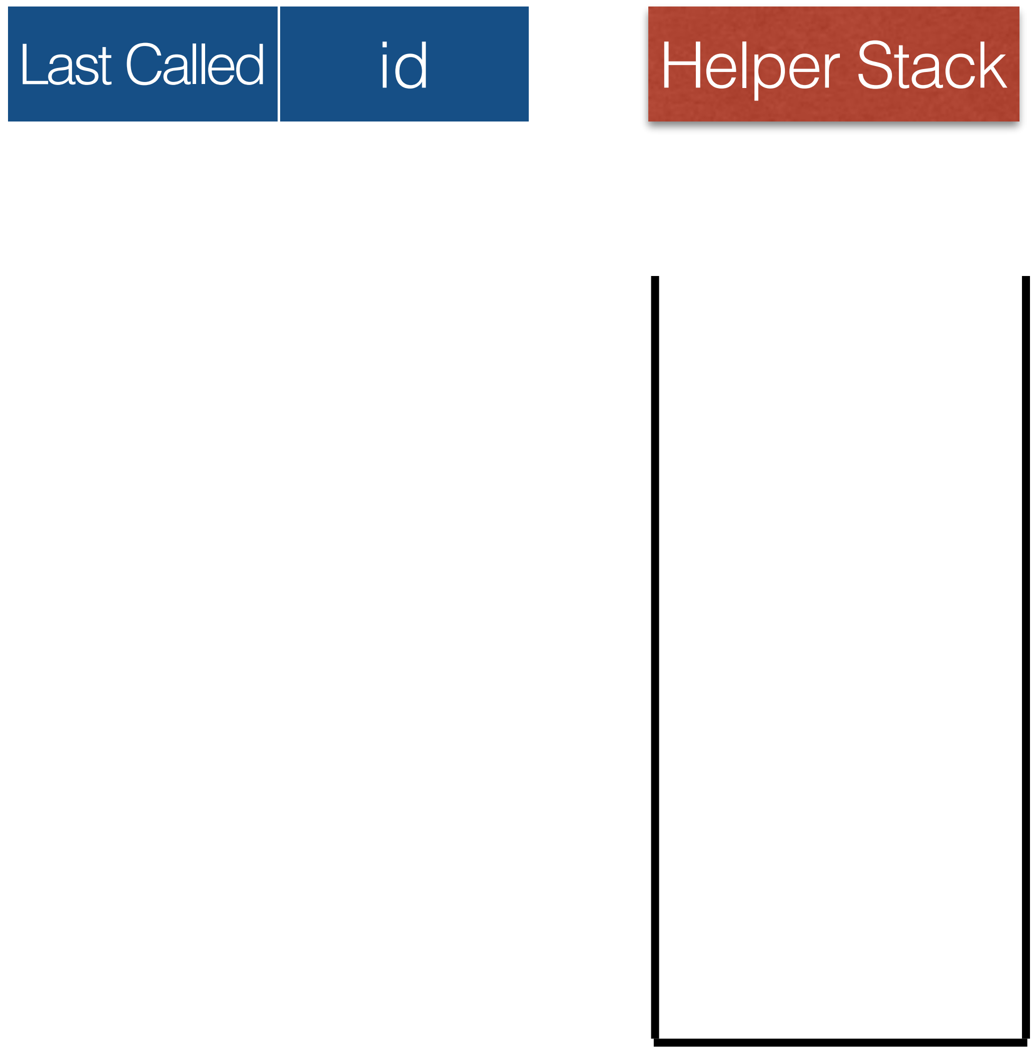
- Use a flag "onstack" to indicate if there is an unencoded call edge in current sub-path.
- If the encoding id of a sub-path is bigger than maxID, then adjust id=id-(maxID+1) and set onstack=true.
- In each decoding iteration:

  1) If id=0 and onstack=true (i.e. id=maxID+1), then try to match the decoded context with the saved encoding context on the top of helper stack.

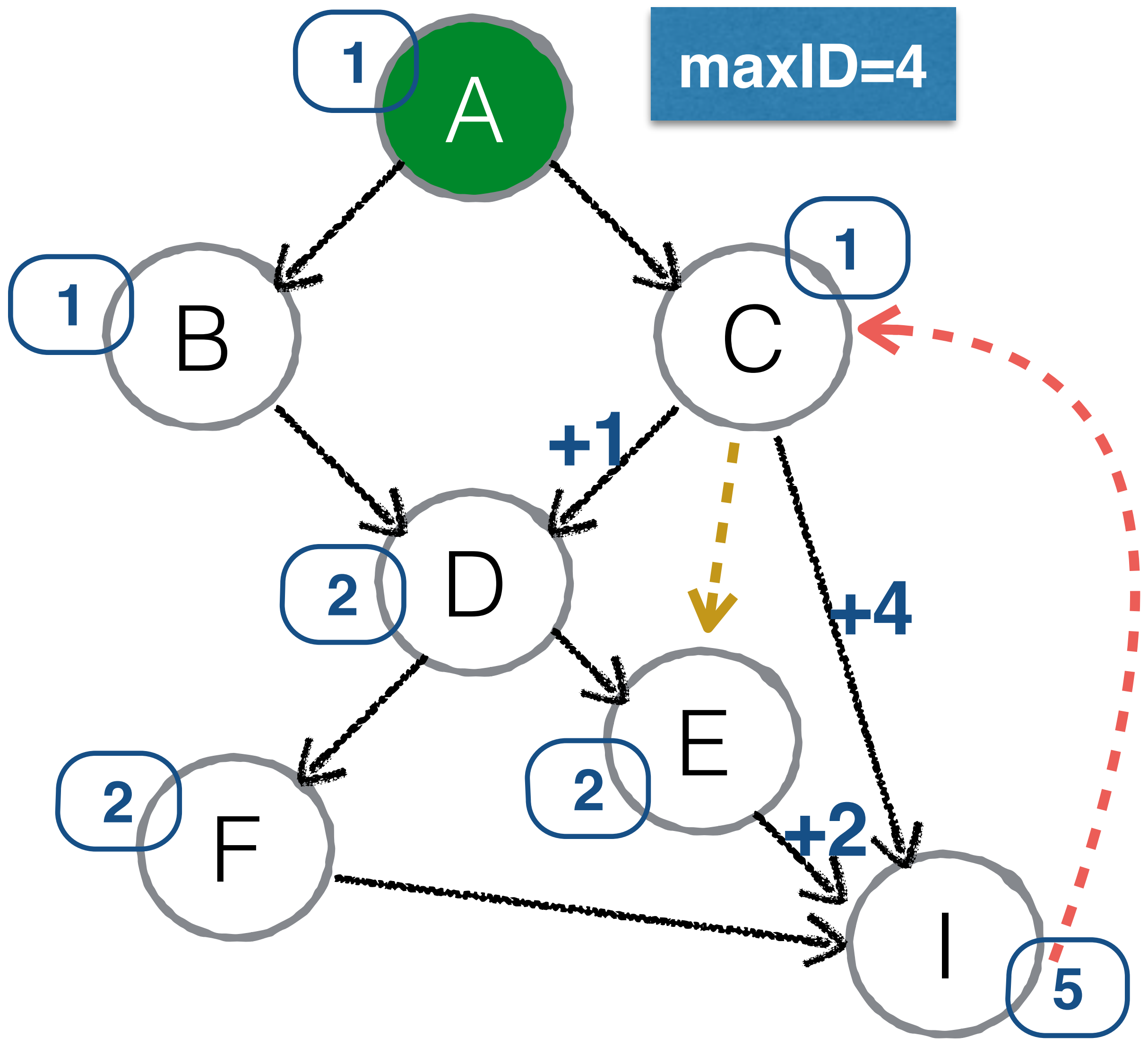
  2) Decode the acyclic sub-path.

Encoding Example



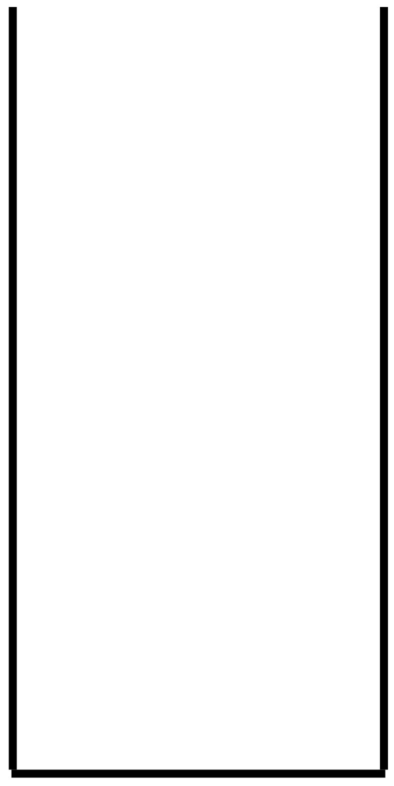


Encoding Example

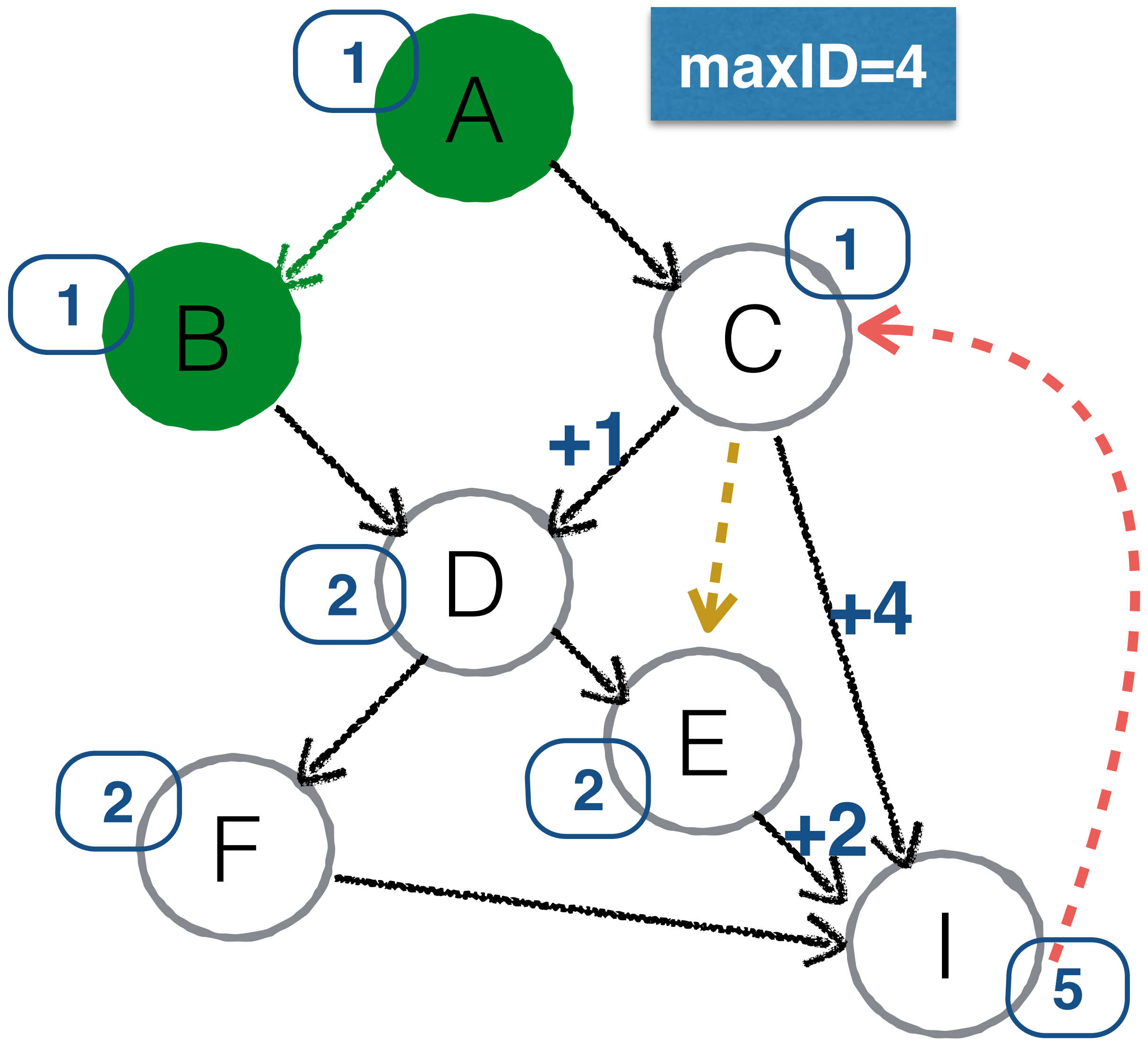


Last Called	id
A	0

Helper Stack

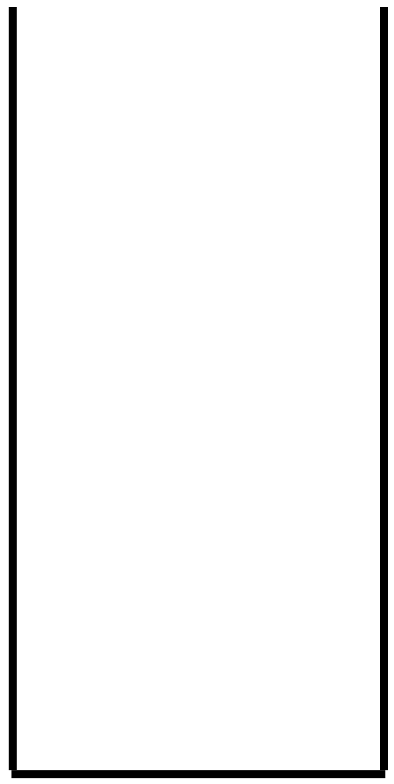


Encoding Example

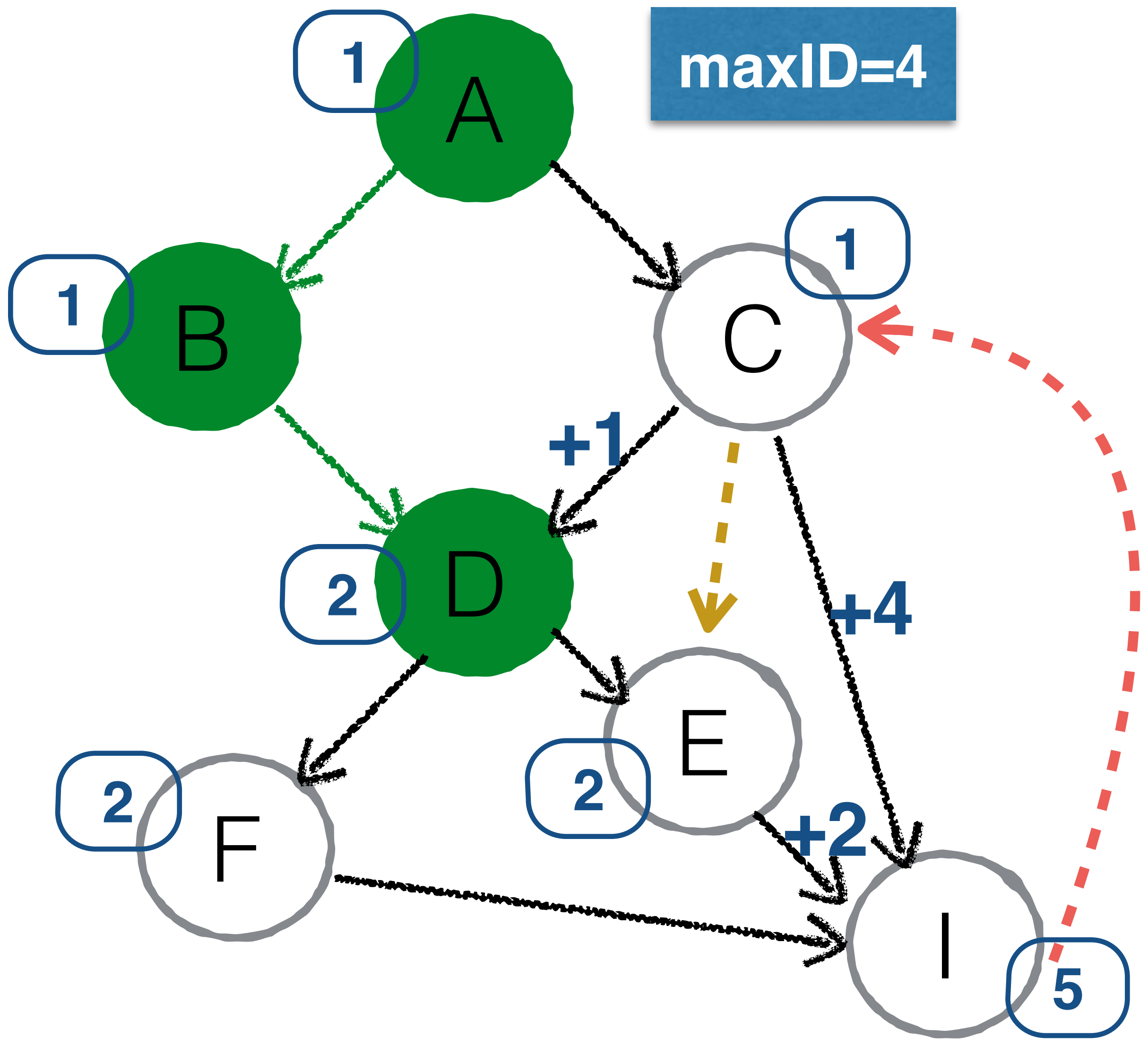


Last Called	id
A	0
B	0

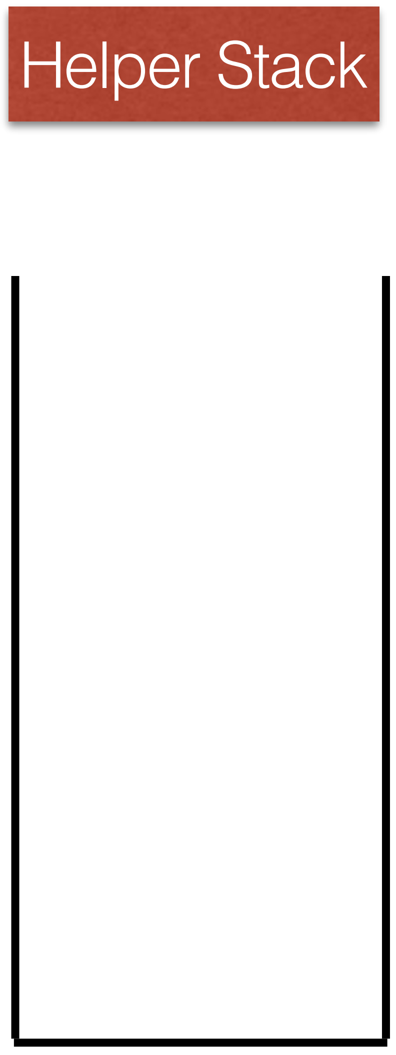
Helper Stack



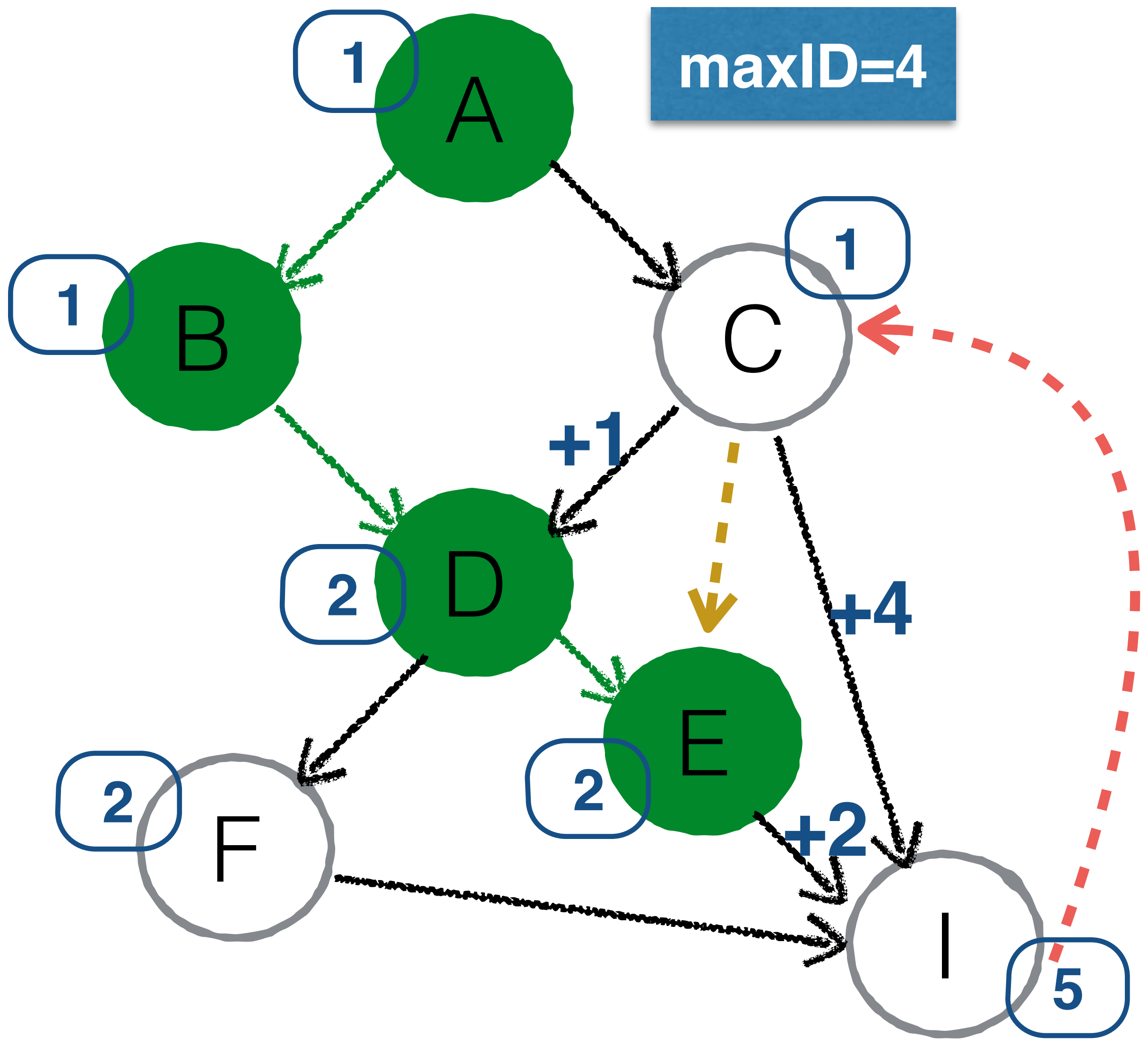
Encoding Example



Last Called	id
A	0
B	0
D	0

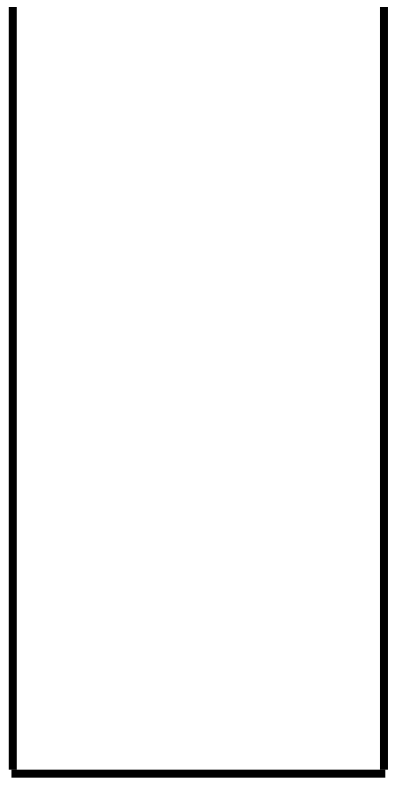


Encoding Example

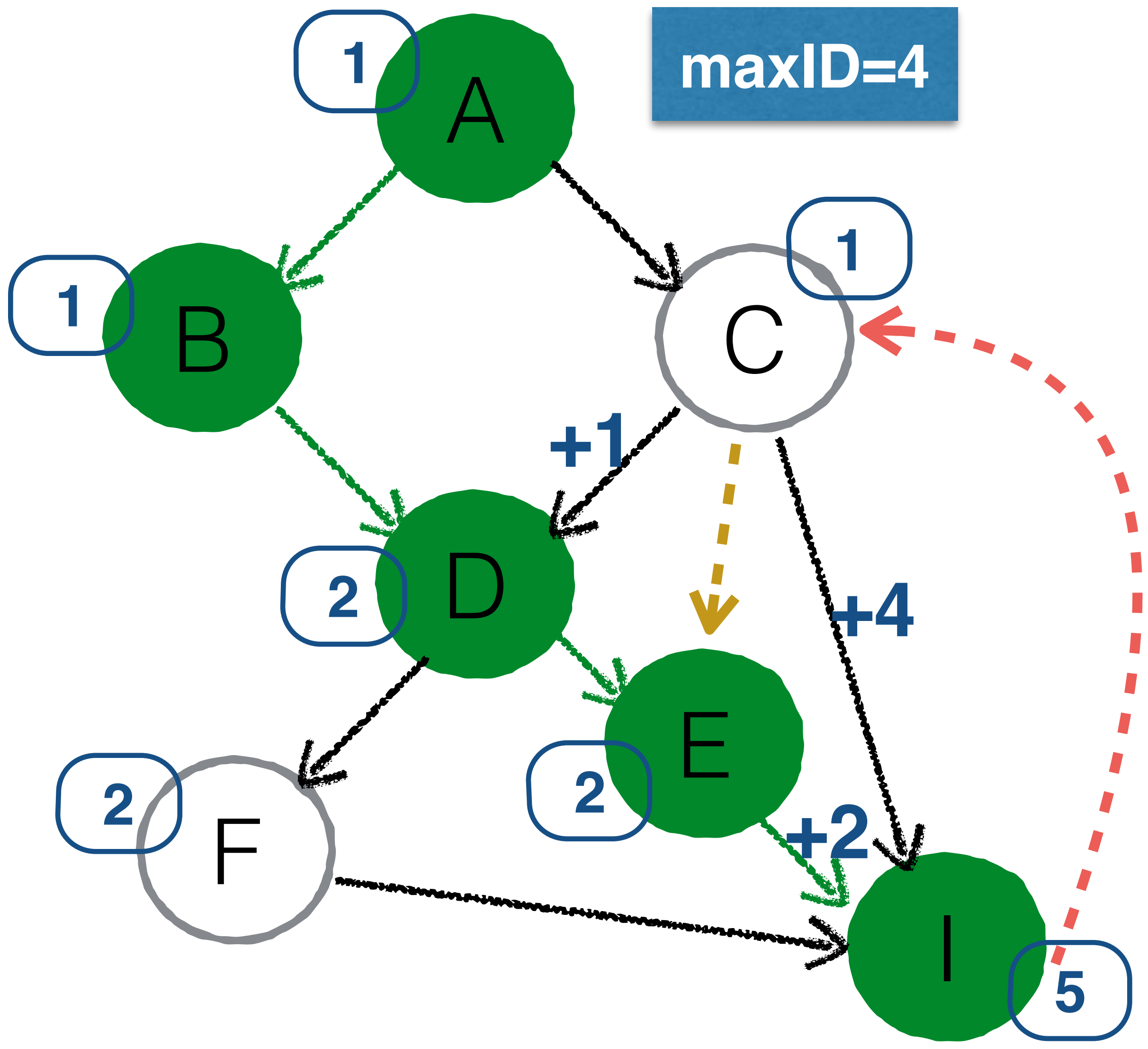


Last Called	id
A	0
B	0
D	0
E	0

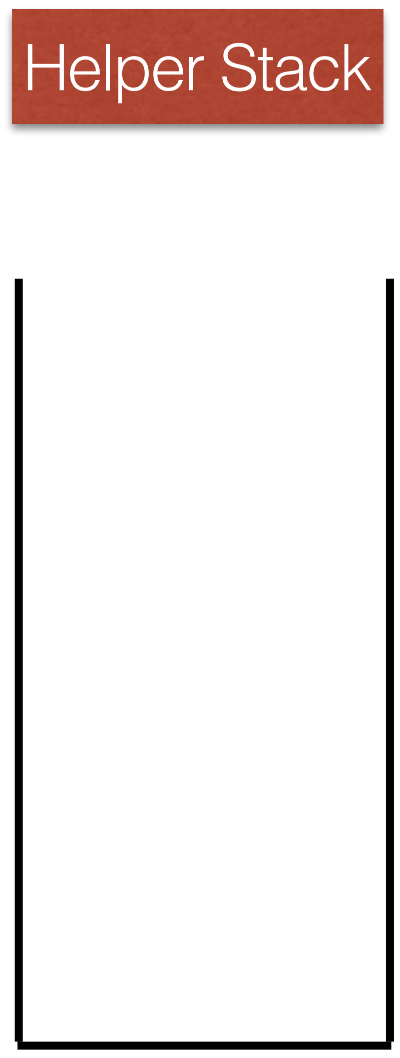
Helper Stack



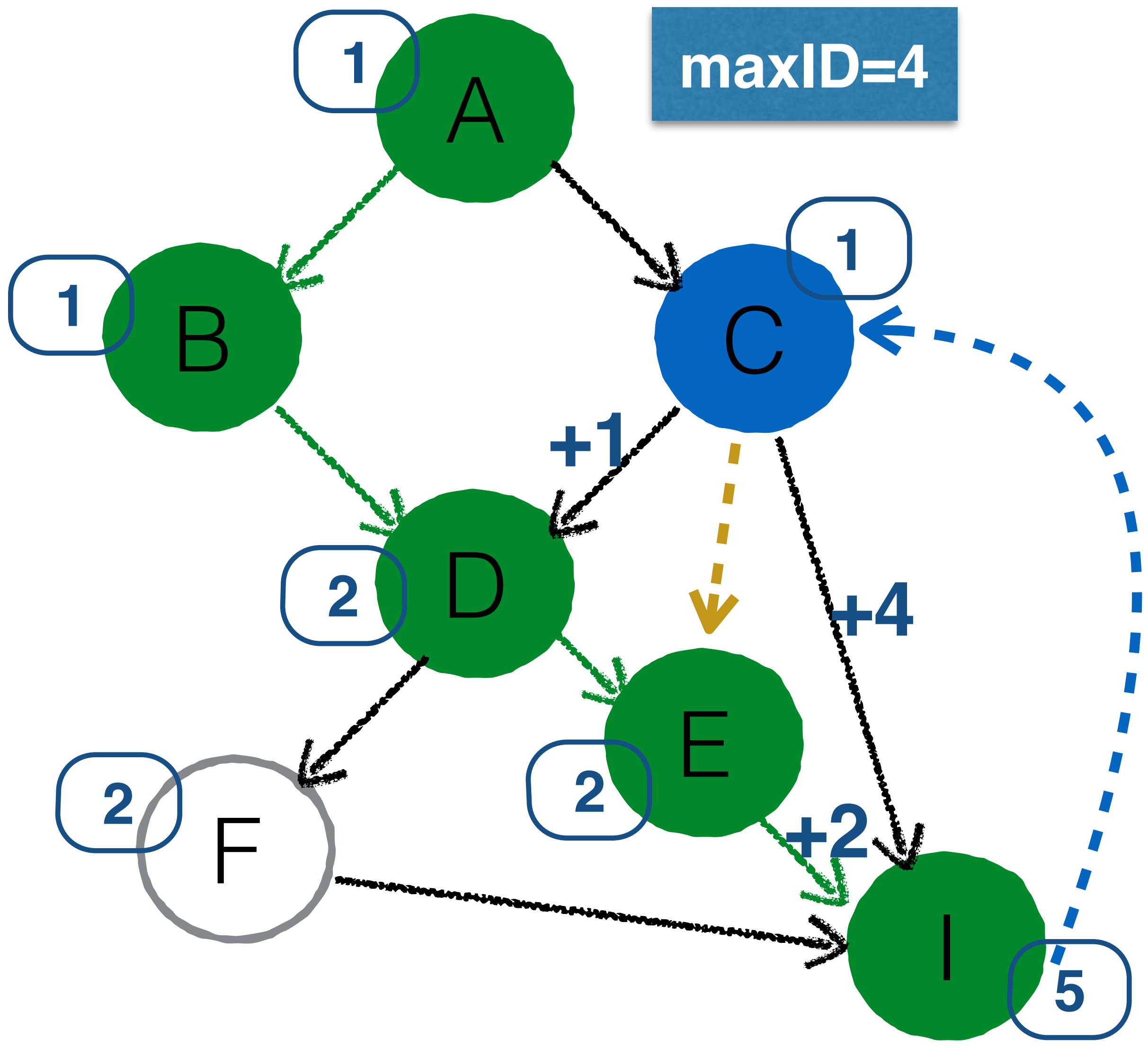
Encoding Example



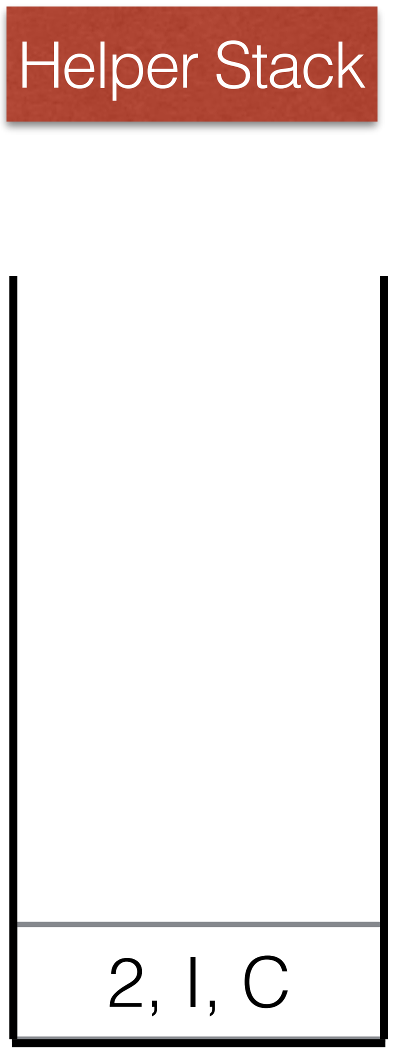
Last Called	id
A	0
B	0
D	0
E	0
I	2



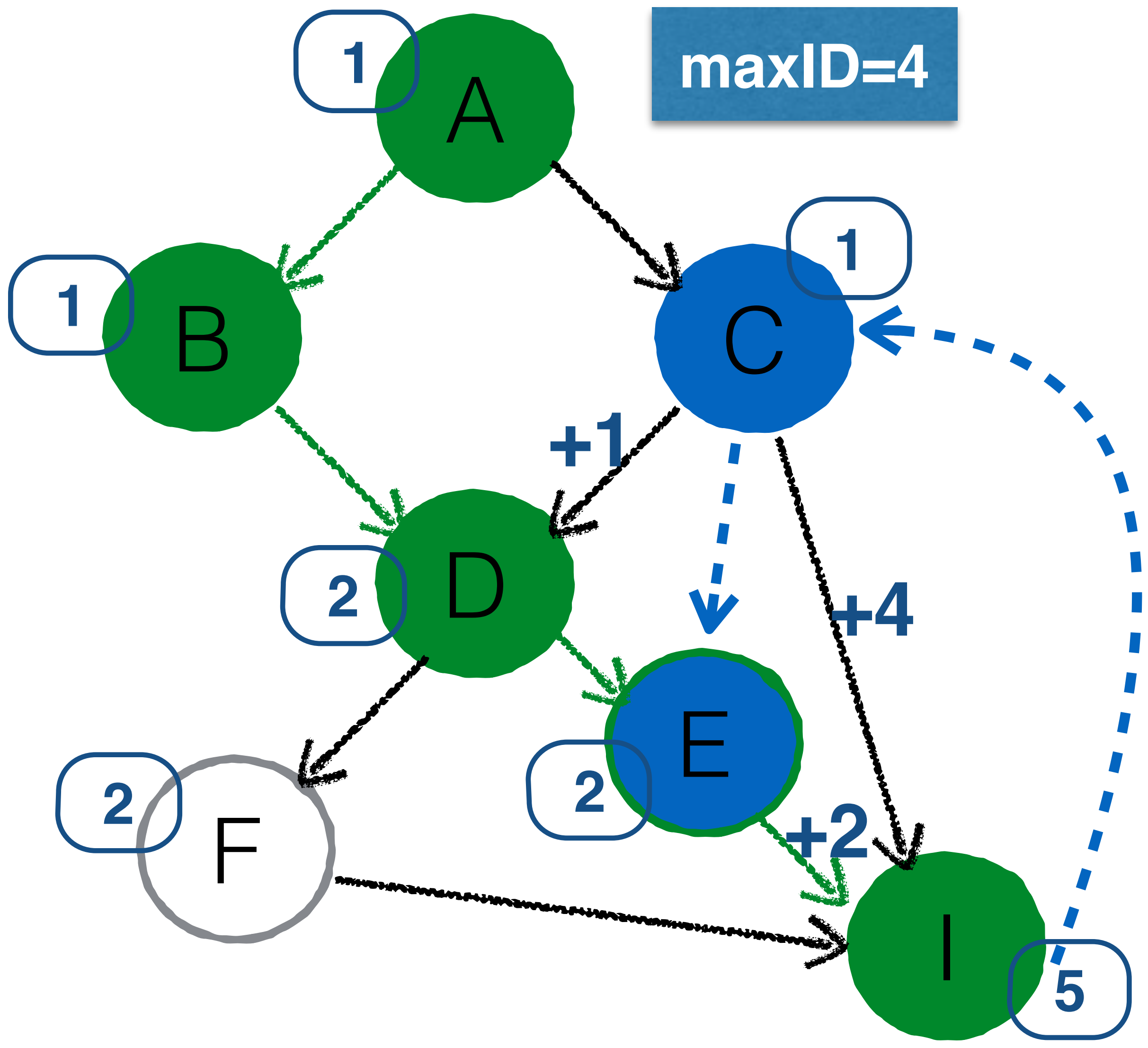
Encoding Example



Last Called	id
A	0
B	0
D	0
E	0
I	2
C	5

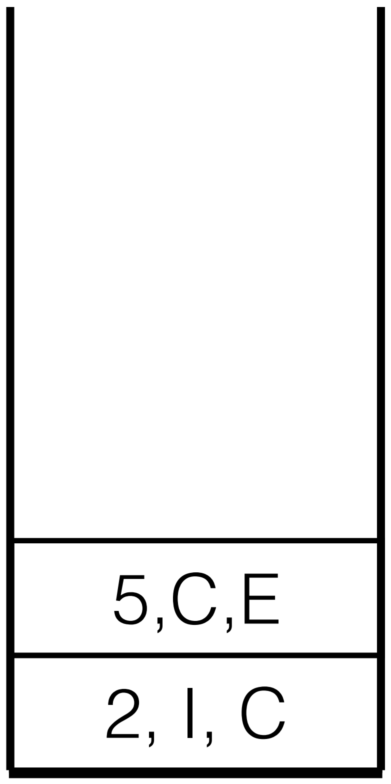


Encoding Example

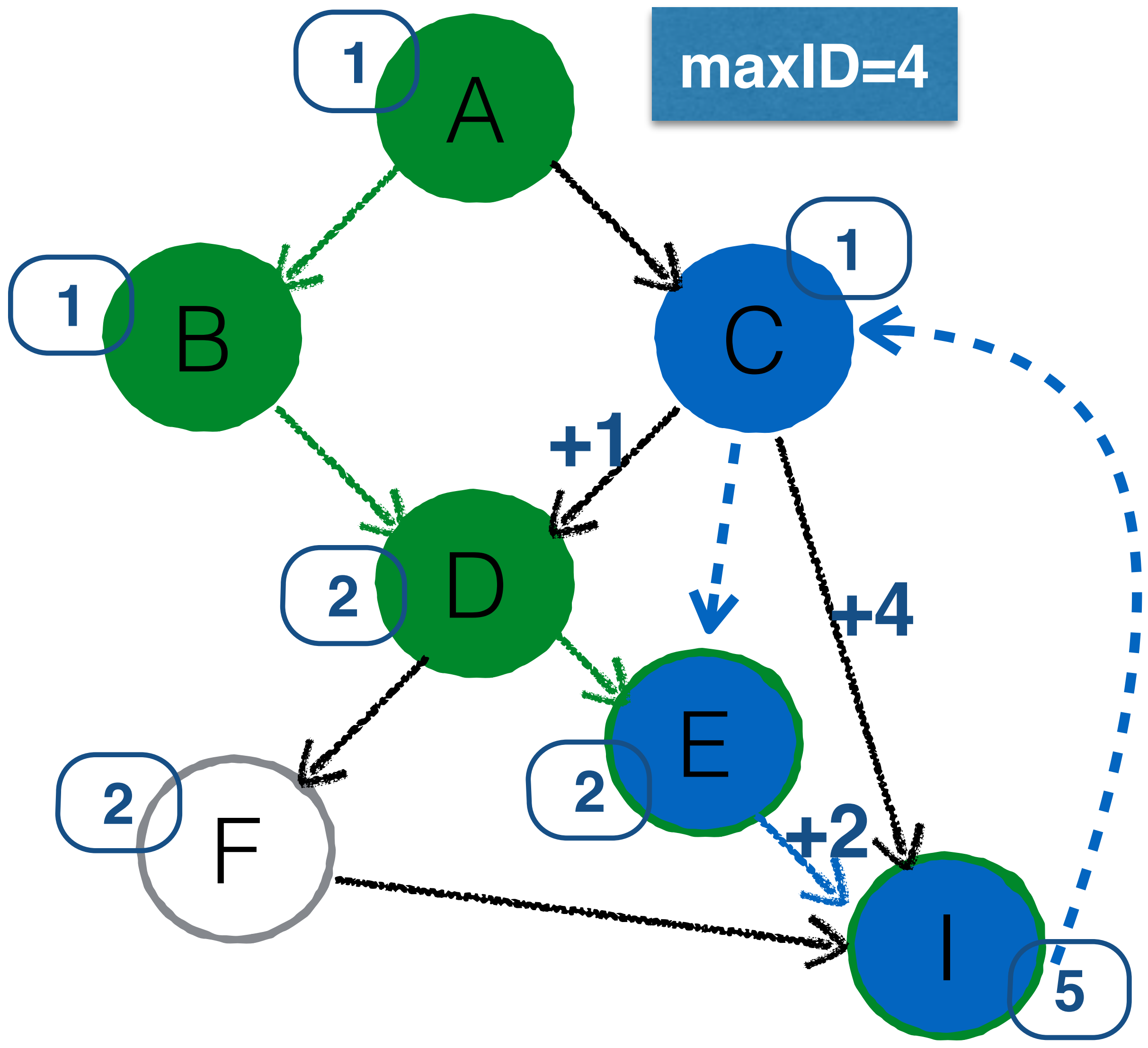


Last Called	id
A	0
B	0
D	0
E	0
I	2
C	5
E	5

Helper Stack

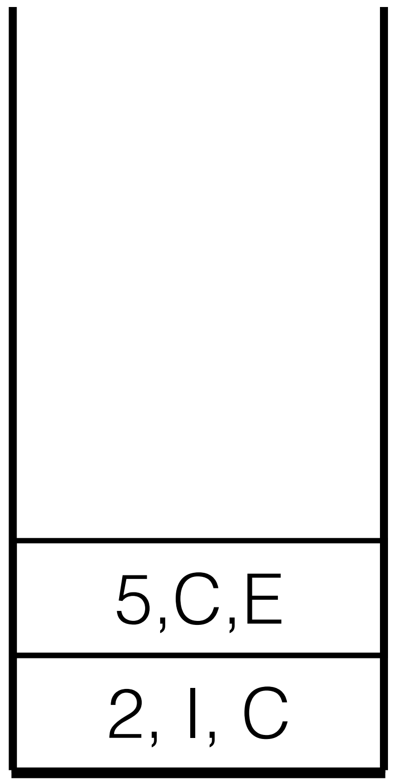


Encoding Example

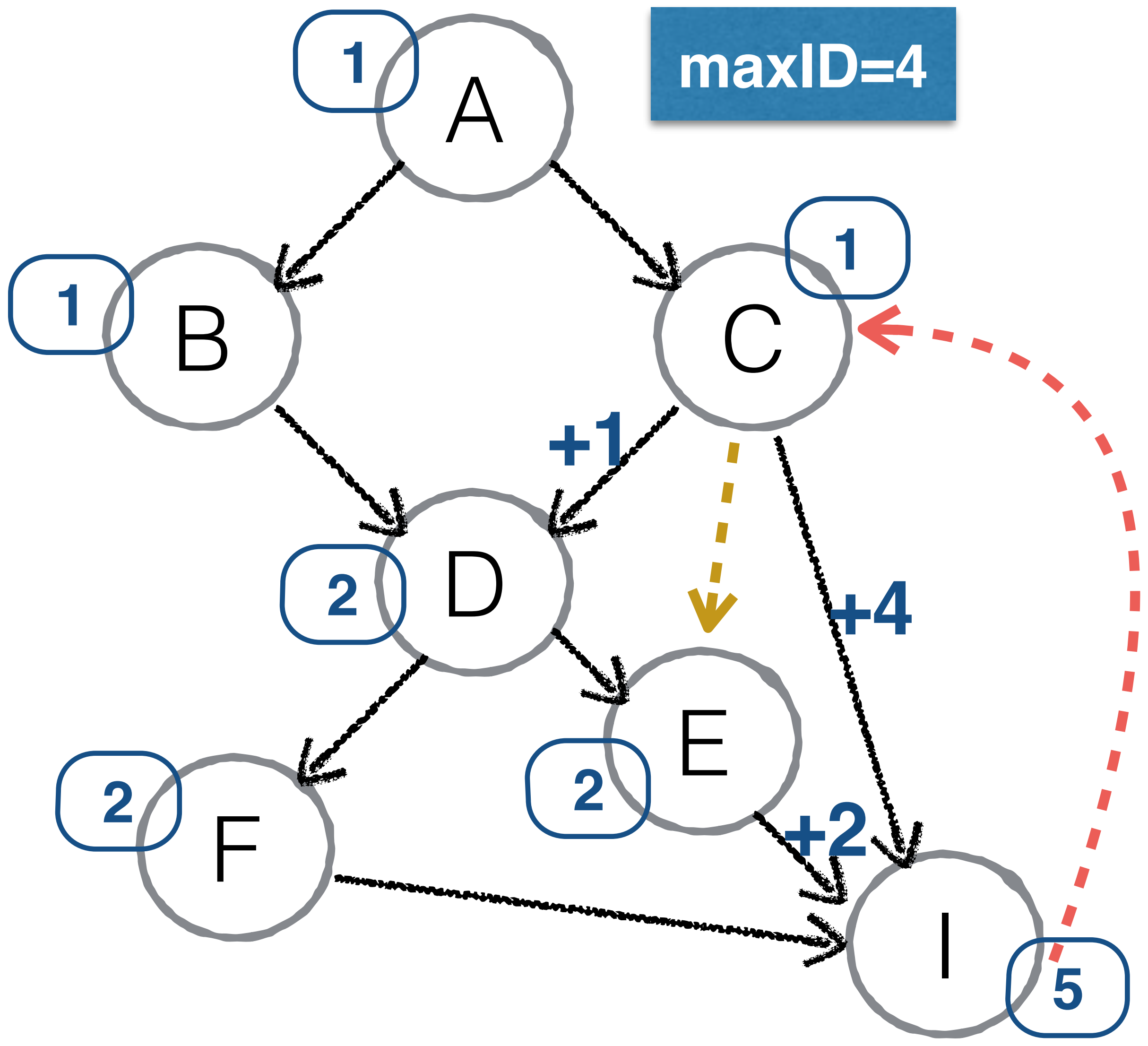


Last Called	id
A	0
B	0
D	0
E	0
I	2
C	5
E	5
I	7

Helper Stack



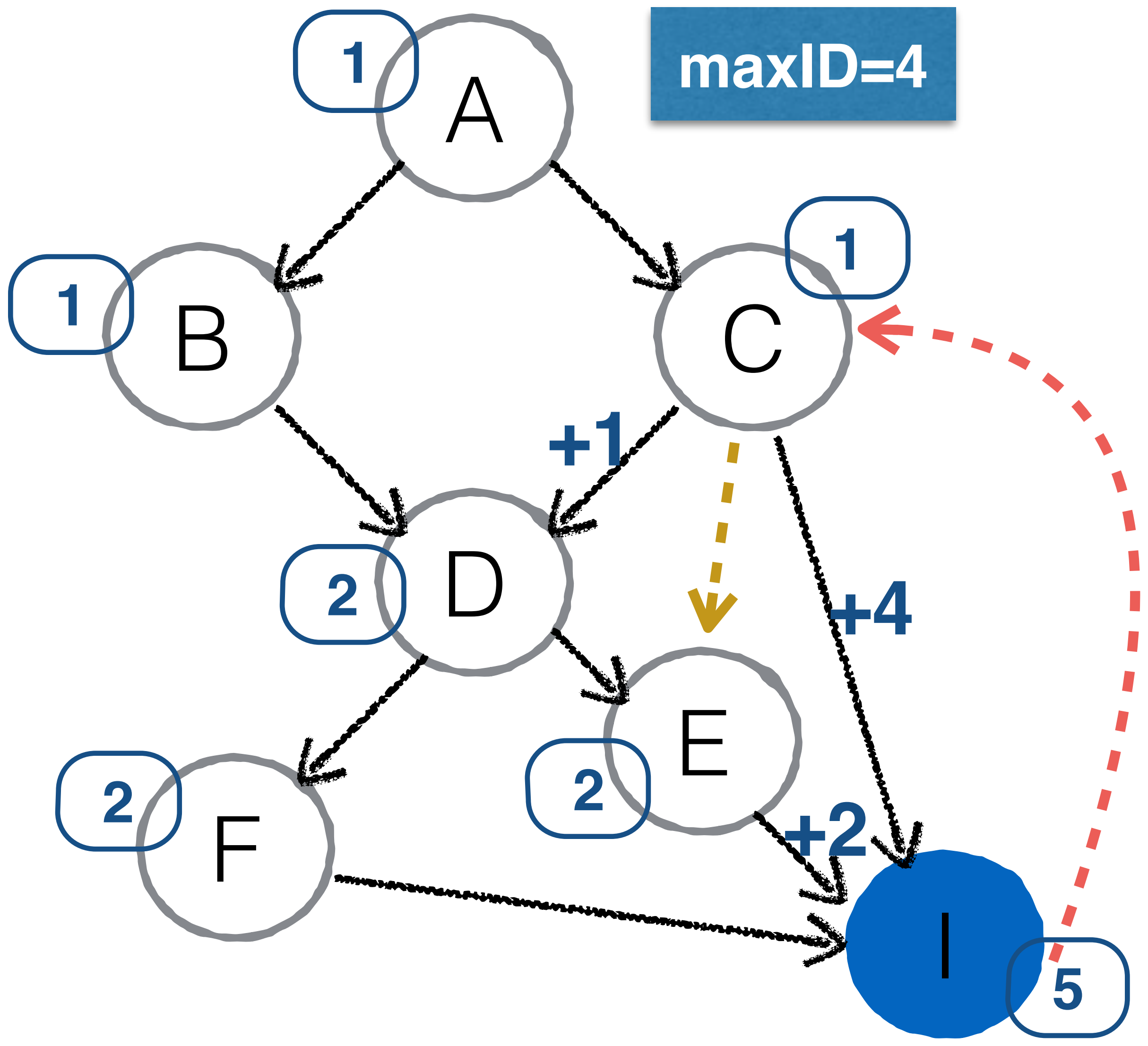
Decoding Example



Encoding result:
pc in function **I**, id=**7**

5,C,E
2, I, C

Decoding Example



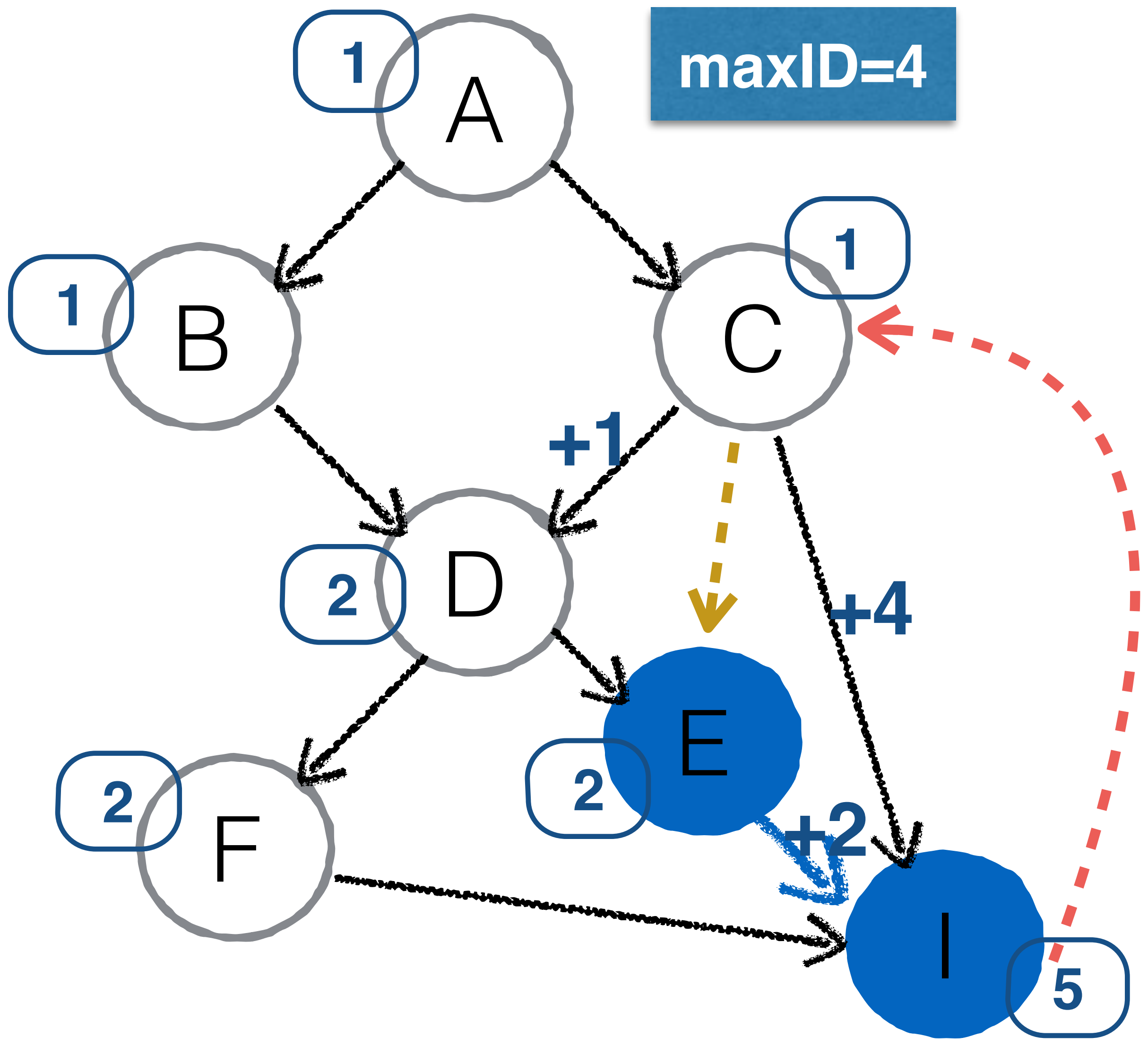
Encoding result:
pc in function **I**, id=**7**

Helper Stack

5,C,E
2, I, C

Decoding Intialization:
a) print “I”
b) (id=7) > (maxID=4), so adjust id=id-(maxID+1)=2 and set onstack=ture.

Decoding Example



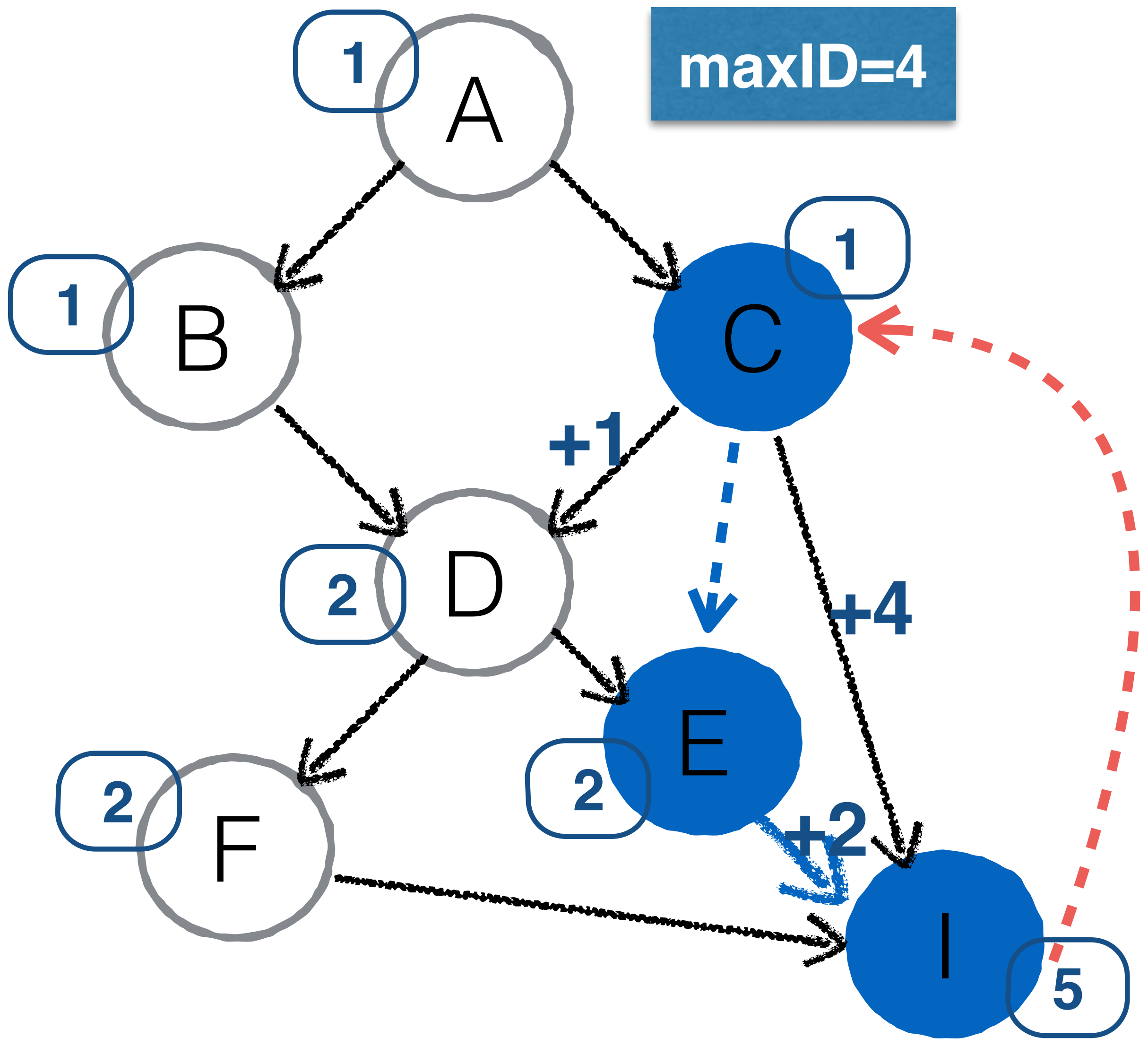
current condition:
pc in function **I**, id=**2,**
onstack=**ture**

Helper Stack

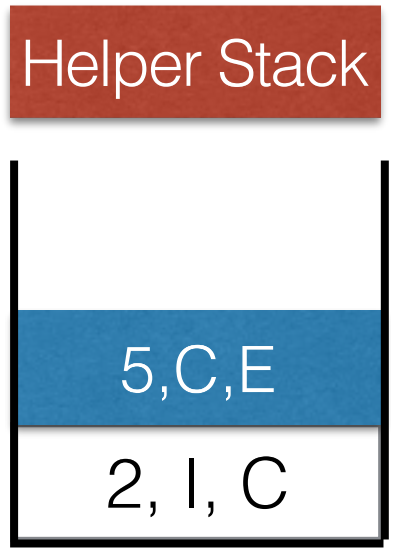
5,C,E
2, I, C

Decoding step 1:
a) Since id!=0, continue decoding current sub-path.
b) Edge EI is decoded, and id = 2-2 = 0.
c) Print “E”.

Decoding Example

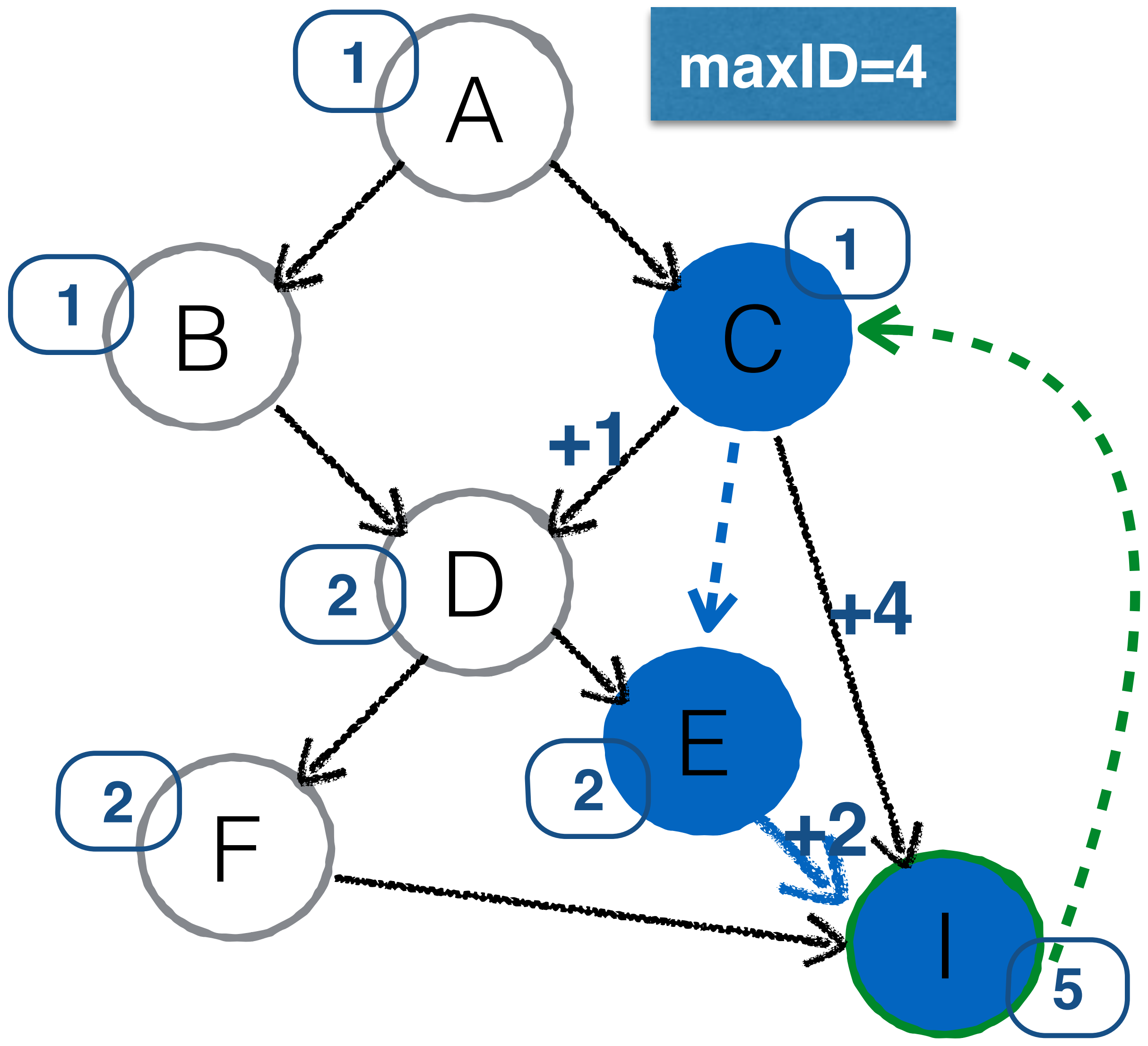


current condition:
pc in function **E**, id=**0,**
onstack=**ture**

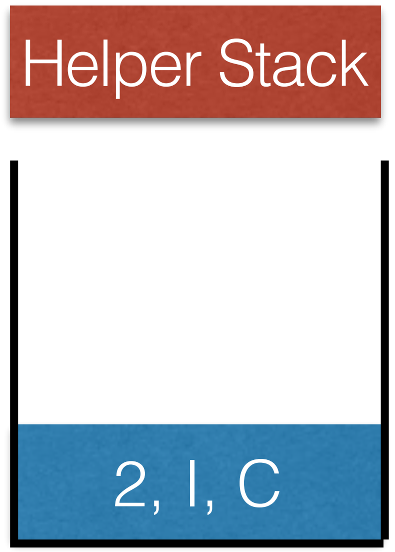


Decoding step 2:
a) Since id=0, onstack=true and the encoding context on the helper stack's top entry matches current context, popup the top entry.
b) Restore current encoding context with the popped encoding context.
c) Print "C".

Decoding Example

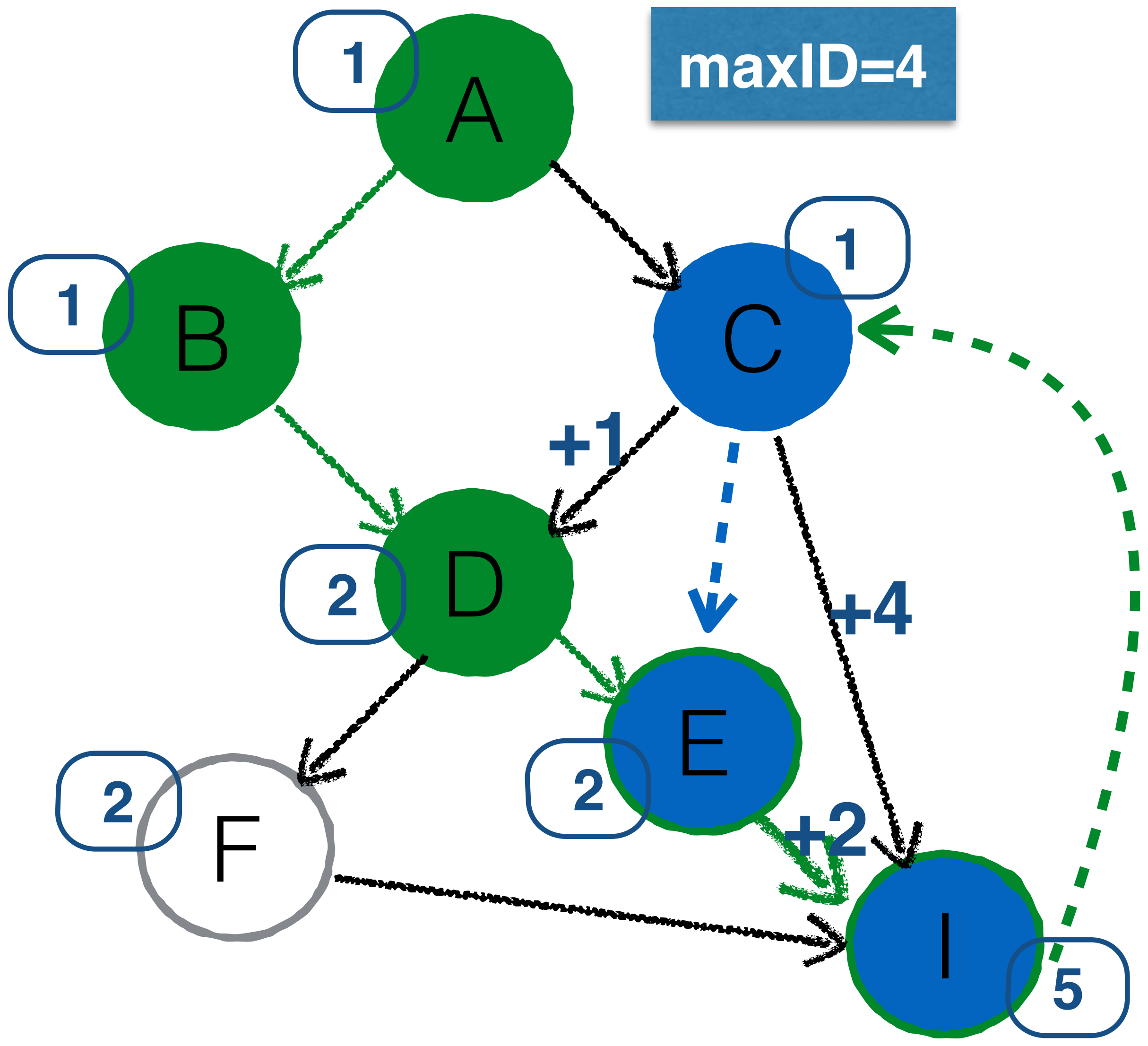


current condition:
pc in function **C**, id=**5,**
onstack=**false.**

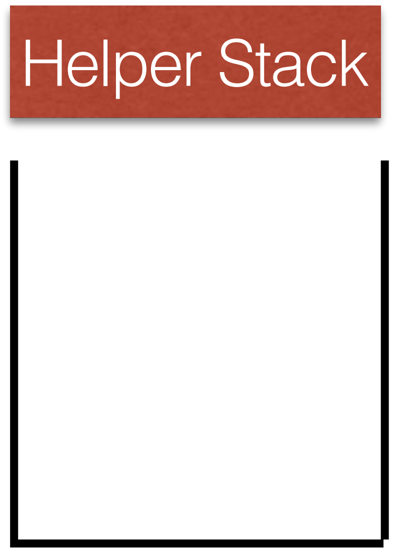


Decoding step 3:
a) (id=5)>(maxID=4), so adjust the value of id=id-(maxID=1)=0 and set onstack=true.
b) Since id=0, onstack=true and the encoding context on the helper stack's top entry matches current context, popup the top entry.
b) Restore current encoding context with the popped encoding context.
c) Print "I".

Decoding Example

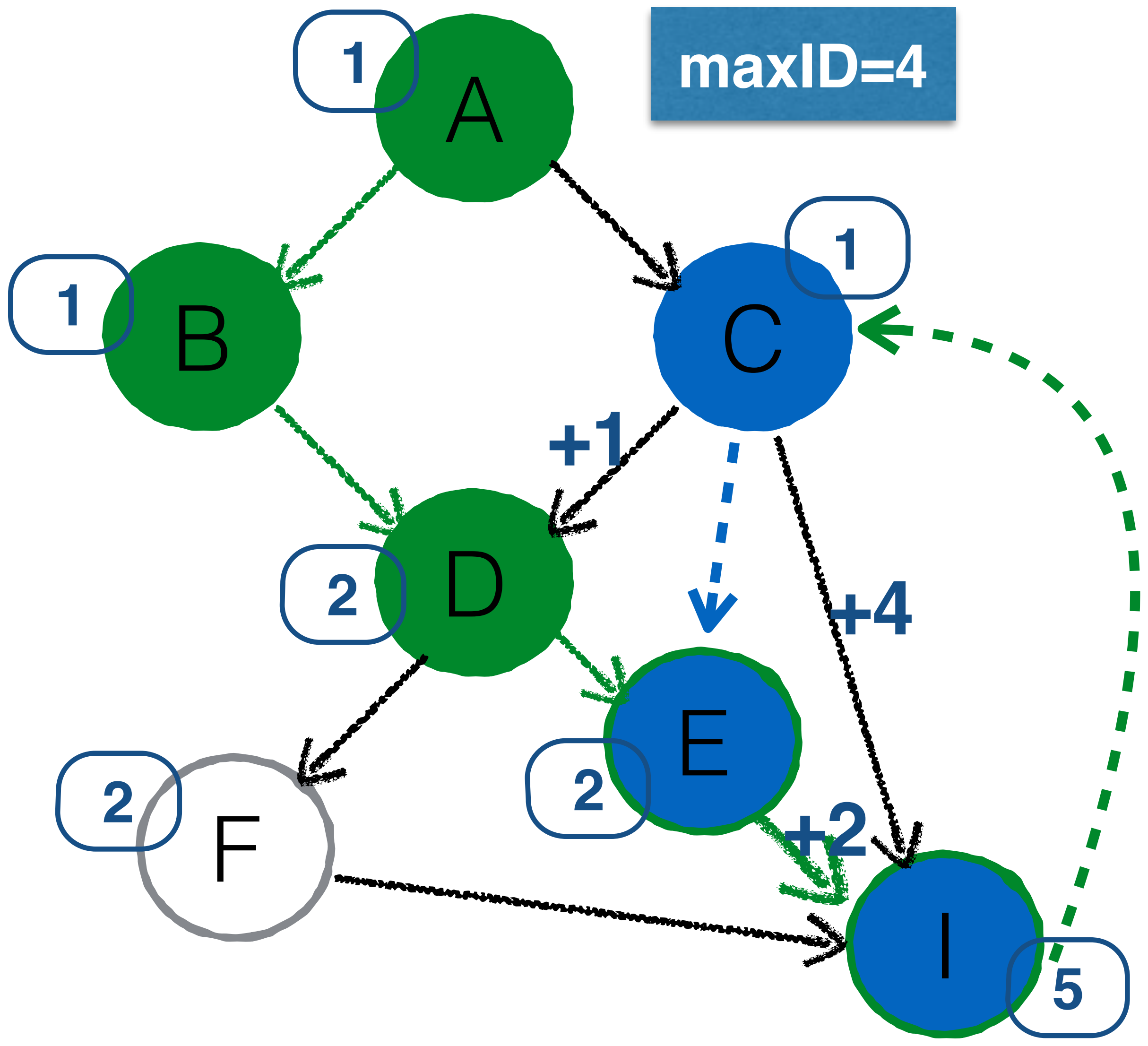


current condition:
pc in function **I**, id=**2,**
onstack=**false.**

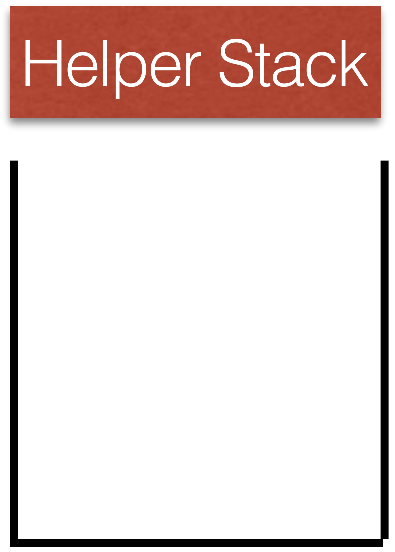


Decoding step 4:
a) Since onstack=false, the acyclic sub-path "ABDEI" is decoded.
b) Print "E", "D", "B", "A".

Decoding Example

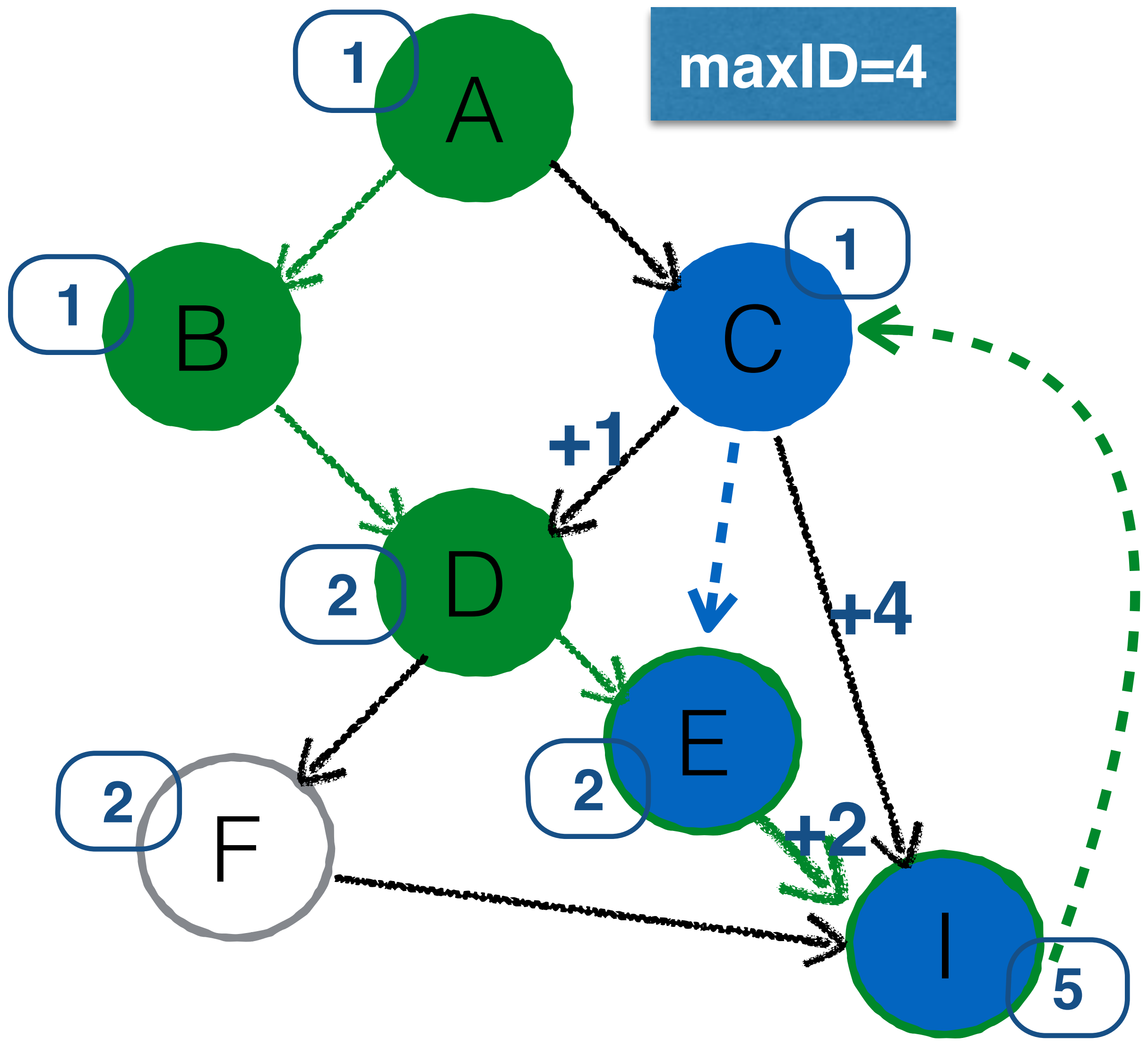


current condition:
pc in function **I**, id=**2,**
onstack=**false.**



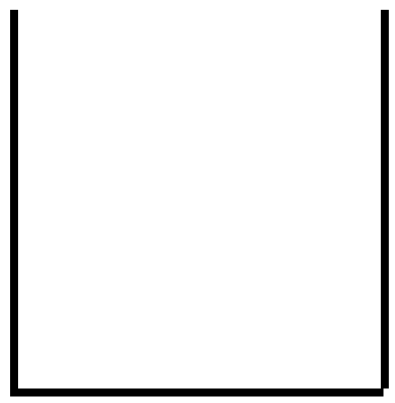
Decoding step 4:
  a) Since onstack=false, the acyclic sub-path “ABDEI” is decoded.
  b) Print “E”, “D”, “B”, “A”.

Decoding Example



current condition:
pc in function **A**, id=**0,**
onstack=**false.**

Helper Stack



Decoding iteration 5:
a) id=0 and helper stack is empty, so the decoding process terminates.
b) Finally, we get the full path "ABDEICEI".

Evaluation

- Experimental Framework
  - Implemented as a shared library
  - To verify the correctness of DACCE, we periodically collect context ids at runtime. we also capture the calling contexts with a stack-walking method. The contexts obtained by the two methods are cross validated.
- Benchmarks
  - SPEC CPU2006 (*ref* input set)
  - Parsec 2.1 (*native* input set)

Benchmarks

Program	Nodes	Edges	maxID	depth	re-encode	calls/s
400.perlbenc	684	3911	1.4E+11	0.20	23	29205101
401.bzip2	50	109	61	0.05	5	7687097
403.gcc	1931	11518	7.0E+13	0.00	110	14710894
429.mcf	11	12	3	0.01	2	295581
445.gobmk	1378	4808	2.4E+11	2.47	76	1335556
… …						
483.xalancb	2170	7321	1422838	6.01	27	25341805
410.bwaves	82	164	73	0.01	6	263845
416.gamess	362	2017	112645	0.03	19	3390329
… …						
447.dealII	792	3369	1132	0.06	47	19533456
450.soplex	225	453	367	0.07	7	312430
453.povray	548	2201	548645	0.76	6	34335309
… …						
blackschole	3	5	5	0.00	11	14646244
bodytrack	218	894	667	0.01	5	6928160
… …						
x264	221	1052	2017	0.00	4	23984355

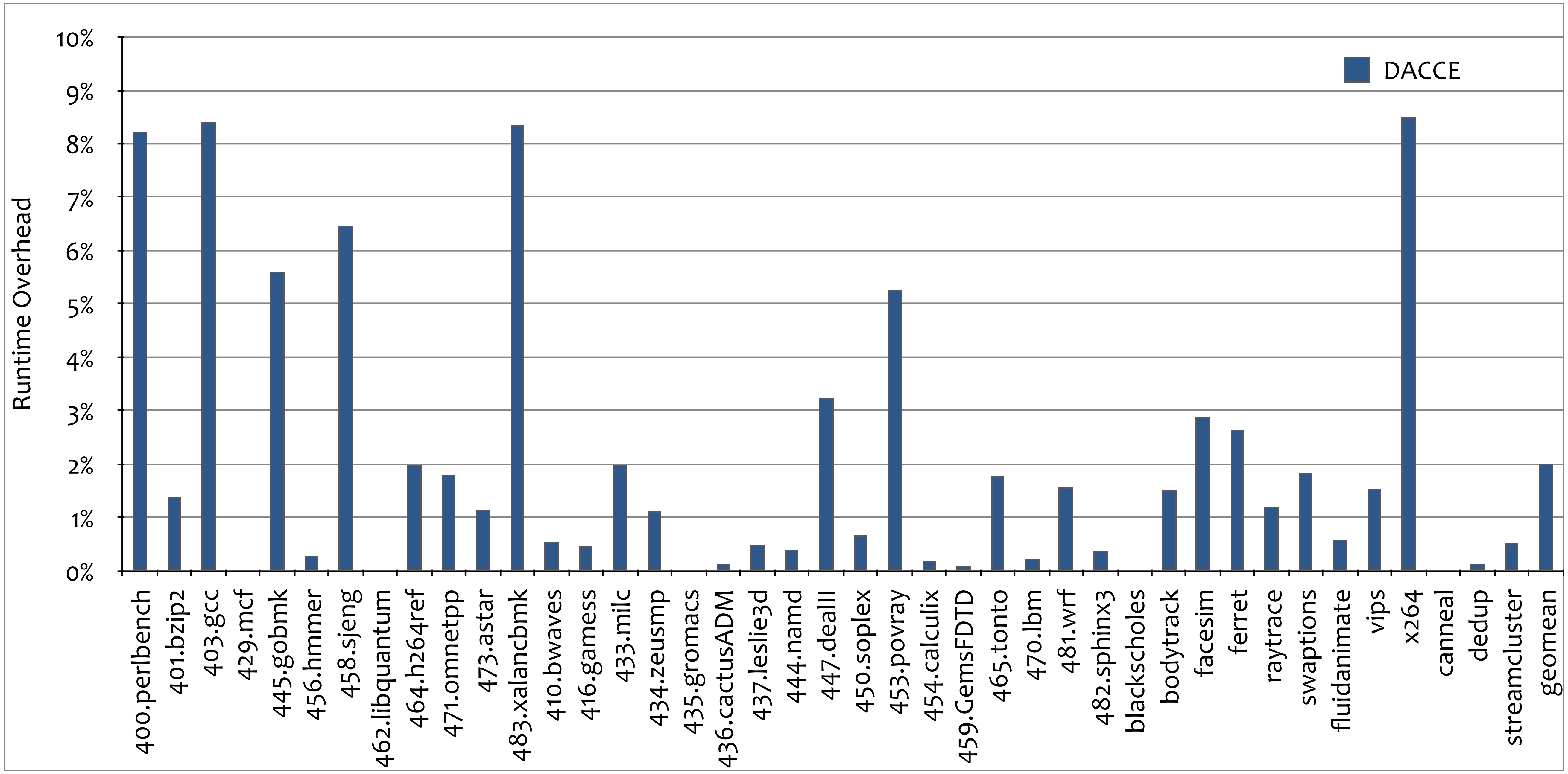
Benchmarks

Program	Nodes	Edges	maxID	depth	re-encode	calls/s
400.perlbenc	684	3911	1.4E+11	0.20	23	29205101
401.bzip2	50	109	61	0.05	5	7687097
403.gcc	1931	11518	7.0E+13	0.00	110	14710894
429.mcf	11	12	3	0.01	2	295581
445.gobmk	1378	4808	2.4E+11	2.47	76	1335556
… …						
483.xalancb	2170	7321	1422838	6.01	27	25341805
410.bwaves	82	164	73	0.01	6	263845
416.gamess	362	2017	112645	0.03	19	3390329
… …						
447.dealII	792	3369	1132	0.06	47	19533456
450.soplex	225	453	367	0.07	7	312430
453.povray	548	2201	548645	0.76	6	34335309
… …						
blackschole	3	5	5	0.00	11	14646244
bodytrack	218	894	667	0.01	5	6928160
… …						
x264	221	1052	2017	0.00	4	23984355

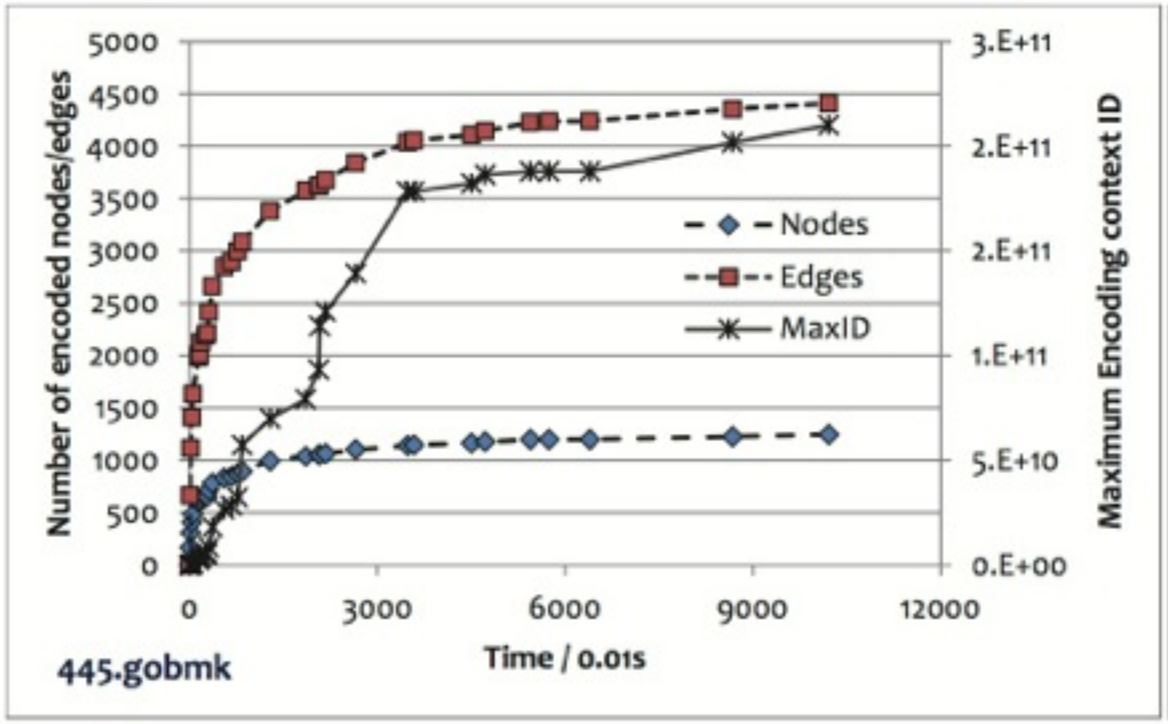
Benchmarks

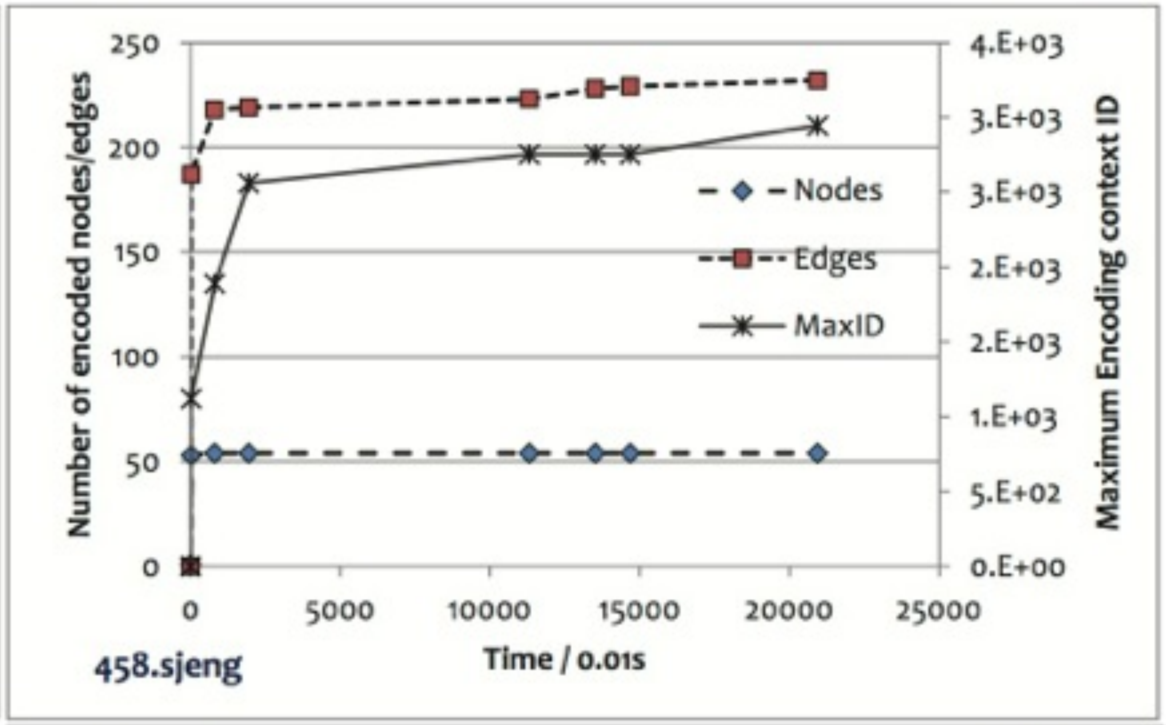
Program	Nodes	Edges	maxID	depth	re-encode	calls/s
400.perlbenc	684	3911	1.4E+11	0.20	23	29205101
401.bzip2	50	109	61	0.05	5	7687097
403.gcc	1931	11518	7.0E+13	0.00	110	14710894
429.mcf	11	12	3	0.01	2	295581
445.gobmk	1378	4808	2.4E+11	2.47	76	1335556
… …						
483.xalancb	2170	7321	1422838	6.01	27	25341805
410.bwaves	82	164	73	0.01	6	263845
416.gamess	362	2017	112645	0.03	19	3390329
… …						
447.dealII	792	3369	1132	0.06	47	19533456
450.soplex	225	453	367	0.07	7	312430
453.povray	548	2201	548645	0.76	6	34335309
… …						
blackschole	3	5	5	0.00	11	14646244
bodytrack	218	894	667	0.01	5	6928160
… …						
x264	221	1052	2017	0.00	4	23984355

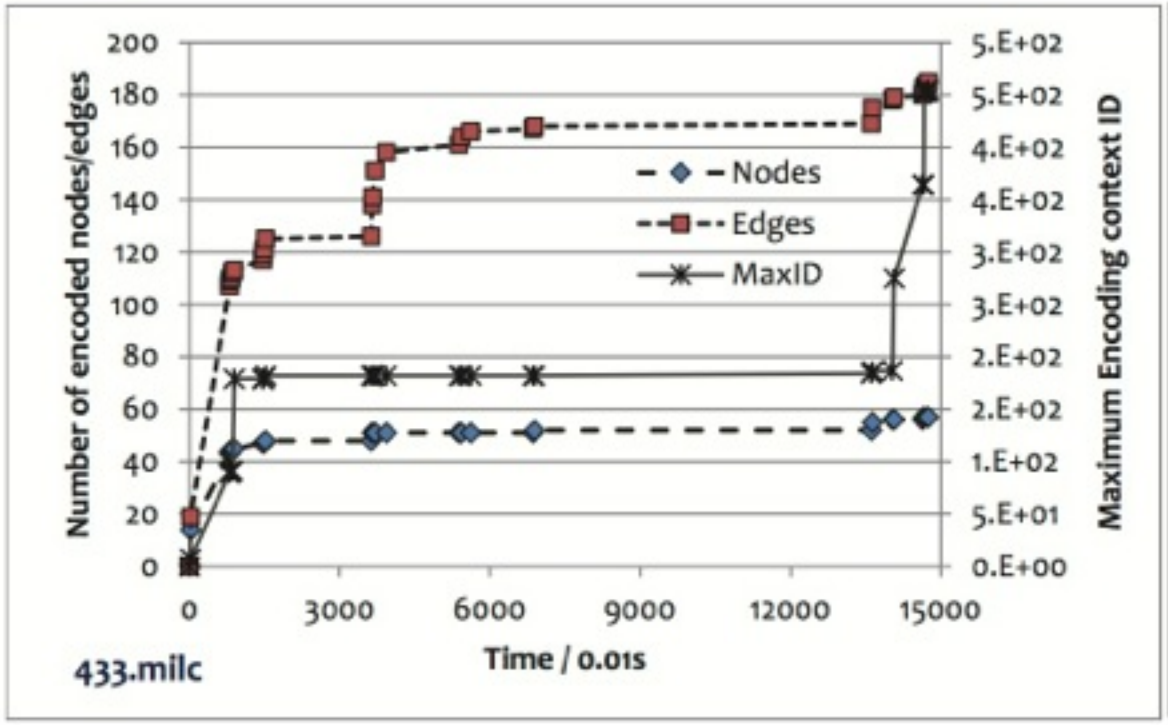
Runtime Overhead

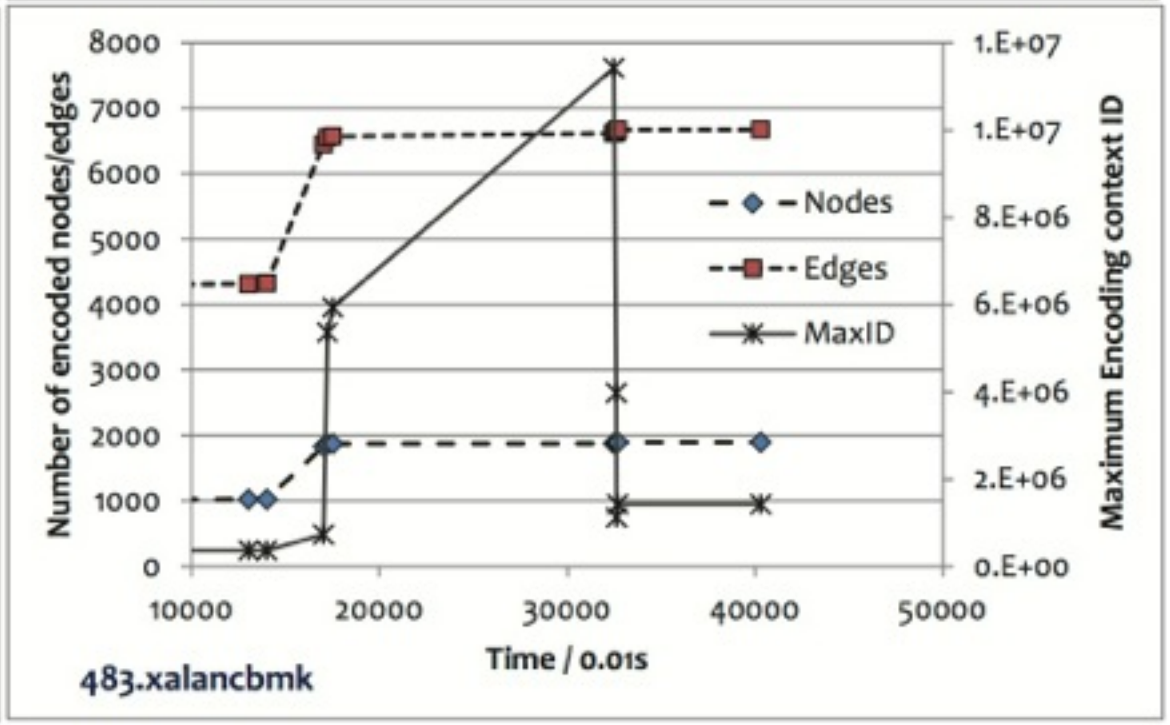


Adaptive Encoding







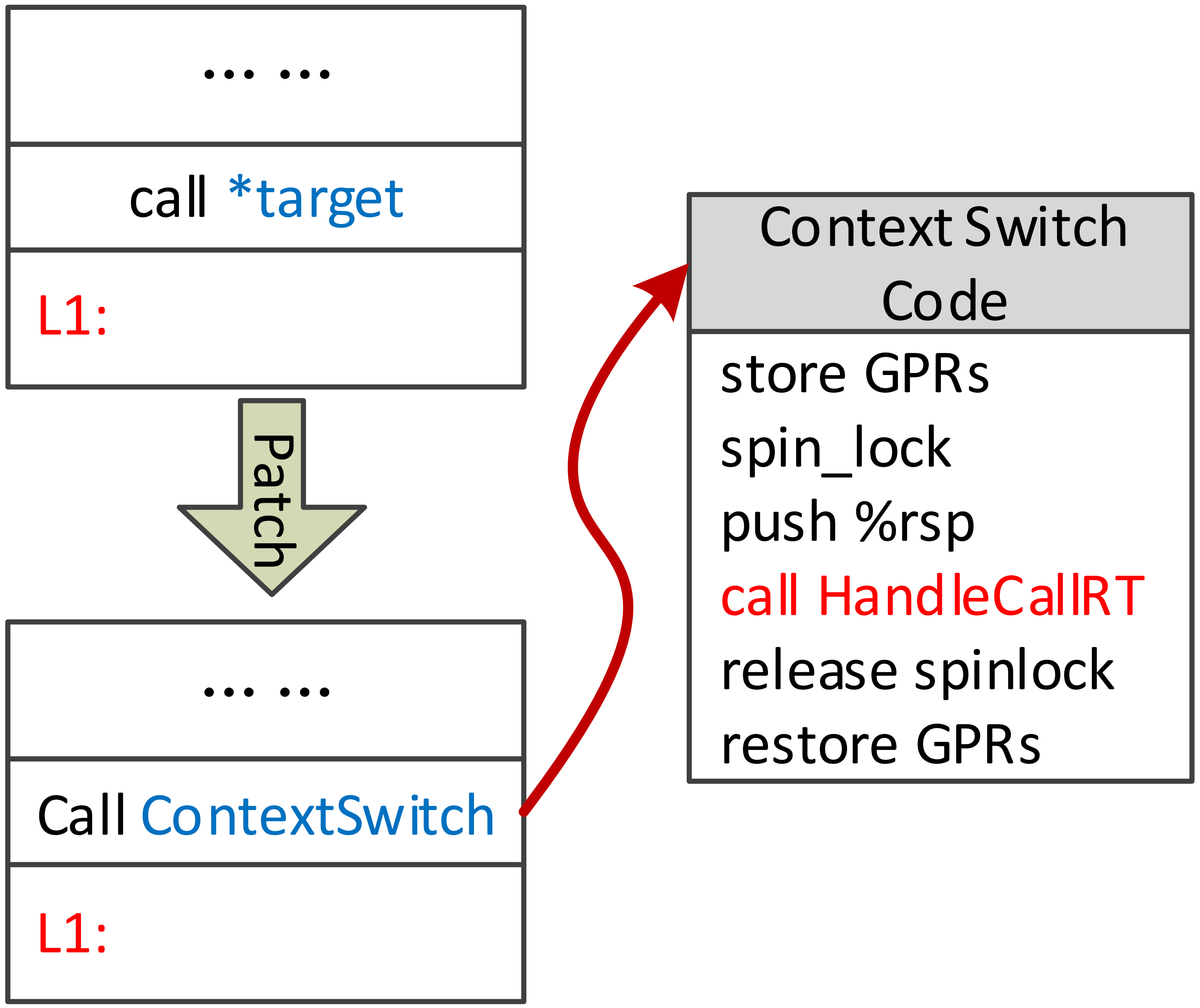


Conclusions

A dynamic and adaptive context encoding algorithm:

- ✓ Does not need extra profiling runs or static program analysis
- ✓ Handle dynamic loadings
- ✓ Adaptive to program behavior changes
- ✓ Efficient in encoding space and time
- ✓ Accurate context information

Thank you &
Questions?


… …
call *target
L1:
Patch
… …
Call ContextSwitch
L1:
Context Switch Code
store GPRs
spin_lock
push %rsp
call HandleCallRT
release spinlock
restore GPRs