



# Σύστημα προανάκτησης σε Blockchain Client με στατική ανάλυση και υποθετική εκτέλεση των Smart Contracts

Διπλωματική Εργασία  
Παναγιώτης Γκόνης

# Ethereum clients

- Συμμετέχοντες στο δίκτυο τρέχουν ένα client
- Κατεβάζουν ολόκληρο το blockchain και εκτελούν όλα τα transaction που περιέχει
- Χρονοβόρα διαδικασία: πολλές ώρες – μέρες σε τυπικό hardware
- Τελική χρήση δίσκου: τάξης TB

# World State

- Σύνολο ζυγών K-V, ίδιο σε όλους τους clients, τροποποιείται από τα Transactions
- Περιέχει λογαριασμούς  
Key=διεύθυνση → Value=λογαριασμός
- Για κάθε συμβολαιο, το χώρο αποθήκευσής του  
Key=slot → Value=περιεχόμενο
- Οργάνωση modified Merkle Patricia Trie (MPT)

# Merkle Patricia Trie

- Λειτουργεί ως KV store
- Δενδρική δομή
- Το κλειδί είναι το μονοπάτι από τη ρίζα στον κόμβο
- Ο κόμβος περιέχει την τιμή για το κλειδί αυτό



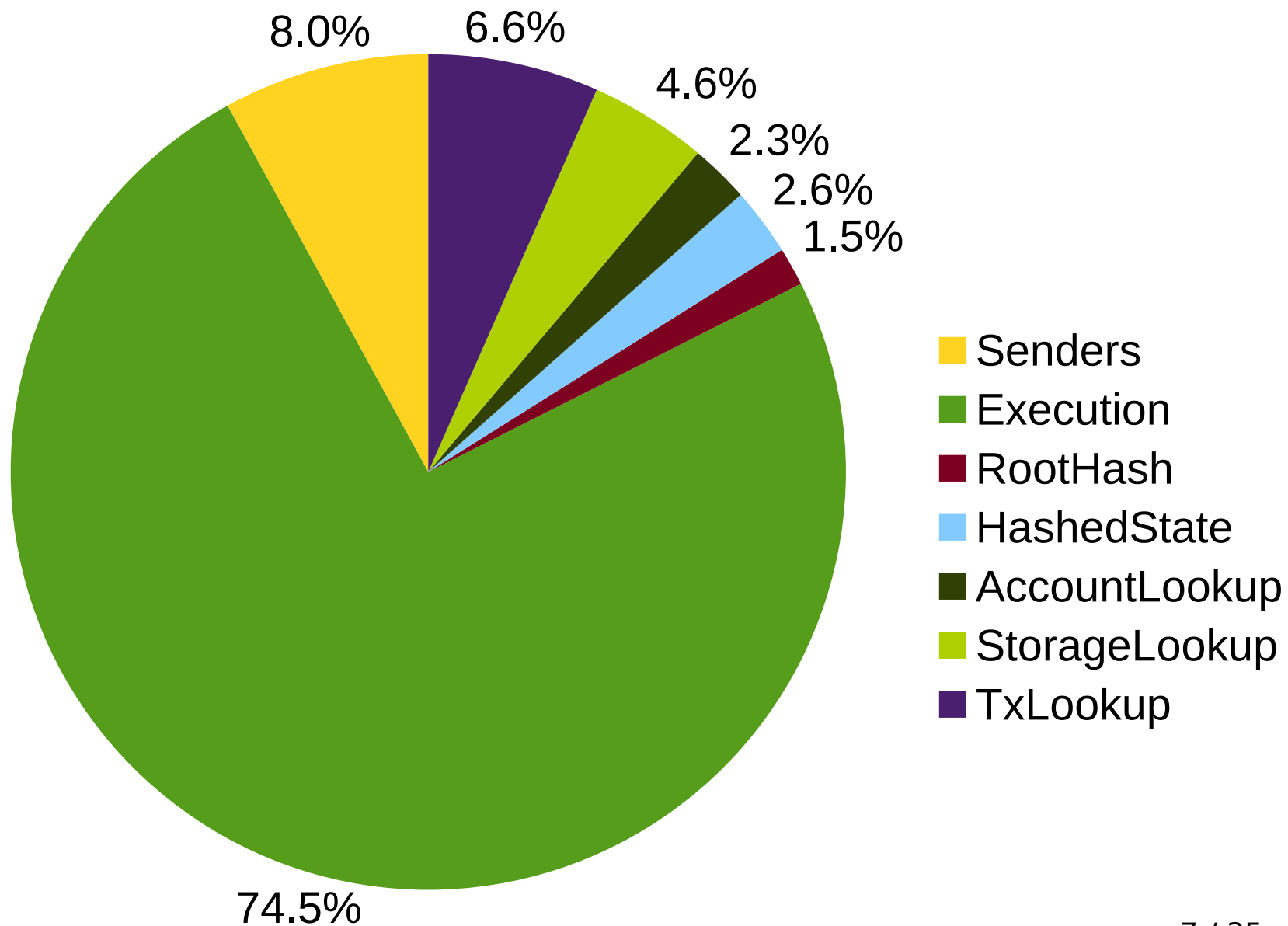
# Go-ethereum

- Official client
- Χρησιμοποιεί εσωτερικά το MPT
- Το αποθηκεύει σε LevelDB
- Trivial αναδρομή σε ιστορικά state
- Μέγεθος βάσης σε archive node: 9 TB

# Erigon

- Fork του go-ethereum
- Η δουλειά χωρίζεται σε stages, πχ download, execute, ...
- Δεν χρησιμοποιεί Trie, αντιθέτως αποθηκεύει σε “flat” μορφή
- Database: MDBX (fork της LMDB), mmap
- Σημαντική βελτίωση σε χρόνο και χώρο
- Μέγεθος βάσης σε archive node: 1,5 TB

# Stages (χωρίς δικτύου)



# Execution Stage

- Γίνεται η εκτέλεση των block, transaction και SC
- Σειριακή εκτέλεση, άγνωστα dependencies μεταξύ transaction
- CPU και IO heavy
- Συνεχής πρόσβαση στο World State (DB), χρονοβόρα blocking reads
- Αν γινόταν prefetching, οι σελίδες της βάσης θα ήταν στη μνήμη (FS cache) και τα reads γρήγορα  
→ στόχος της εργασίας



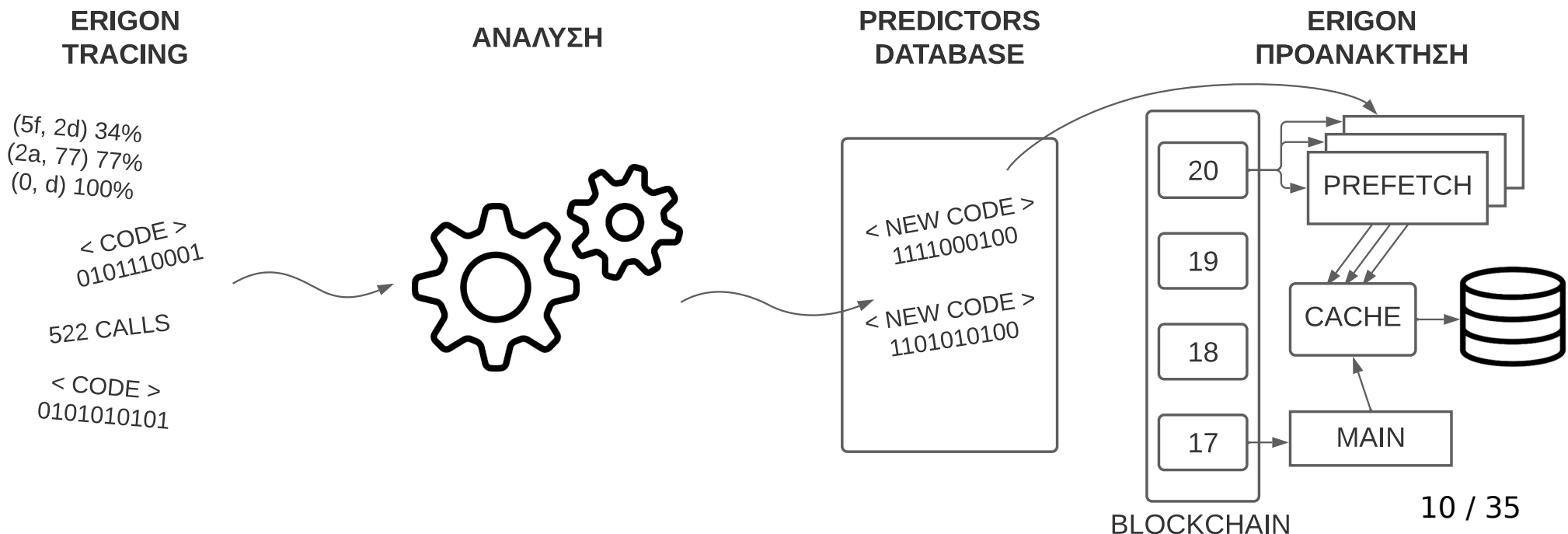
# Πόσο I/O;

- Δοκιμαστική εκτέλεση 30K block, μέτρηση πόσου χρόνου περιμένει στη βάση
  - Με γρήγορο δίσκο NVME: 22 %
  - Με πιο αργό δίσκο sata SSD: 35 %
- Υπάρχει η δυνατότητα για σημαντική βελτίωση

*(με σκληρό δίσκο δεν έγιναν δοκιμές, η ταχύτητα εκτέλεσης είναι 1-2 τάξεις μεγέθους χαμηλότερη)*

# Δομή του συστήματος

- Tracing: Συλλογή μετρικών και κώδικα των SC
- Ανάλυση των SC που συλλέχθηκαν και σύνθεση νέων προγραμμάτων (predictors)
- Prefetching και υποθετική εκτέλεση των predictors

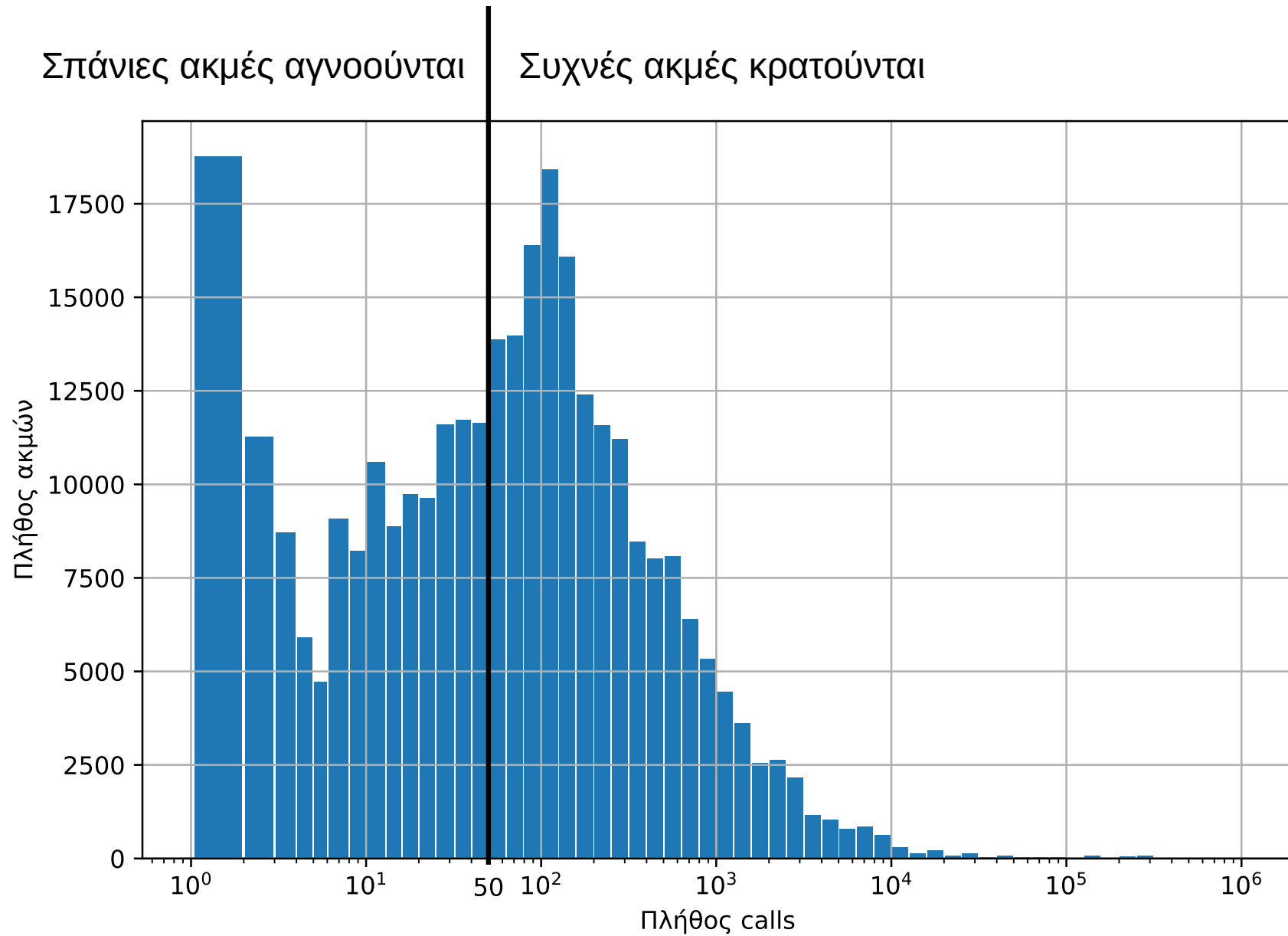


# Tracing

- Κατά την πραγματική εκτέλεση\*, συλλέγουμε τον κώδικα των SC και πλήθος κλήσεων στο καθένα
- Τα δημοφιλή αποθηκεύονται για ανάλυση
- Επίσης συλλέγονται για κάθε JUMP, ο PC πριν και μετά και unique ID του message call
- Τα ζεύγη (ακμές CFG) με “πολλές” κλήσεις δίνονται σαν υποβοήθηση στον αναλυτή

*\*έγινε με δείγμα transaction, offline, αλλά σε μια πραγματική υλοποίηση θα γίνεται online*

# Tracing



# Ανάλυση

- Ξεχωριστή διεργασία, σε Python
- Έχει ως είσοδο SC code, παράγει ως έξοδο προγράμματα (predictors) με παρόμοιο κώδικα
- Οι predictors περιέχουν μόνο “χρήσιμες” εντολές: πρόσβασης storage, calls, κτλ
- Η συμπεριφορά τους επιτρέπεται να διαφέρει από του αρχικού κώδικα
- ευκαιρίες για unsafe optimization, πχ αφαίρεση σπάνιων εντολών

100

- [illegible]

# Ανάλυση 2/12

- Κάνει disassemble

```
00000000: PUSH1 0x80
00000002: PUSH1 0x40
00000004: MSTORE
00000005: PUSH1 0x4
00000007: CALLDATASIZE
00000008: LT
00000009: PUSH2 0x56
0000000c: JUMPI
0000000d: PUSH4 0xffffffff
00000012: PUSH29 0x1000000000000000...
00000030: PUSH1 0x0
00000032: CALLDATALOAD
00000033: DIV
00000034: AND
. . . .
```

# Ανάλυση 3/12

- Το χωρίζει σε basic blocks

```
----- BLOCK ~0 -----  
00000000: PUSH1 0x80  
00000002: PUSH1 0x40  
00000004: MSTORE  
00000005: PUSH1 0x4  
00000007: CALLDATASIZE  
00000008: LT  
00000009: PUSH2 0x56  
0000000c: JUMPI
```

```
----- BLOCK ~d -----  
0000000d: PUSH4 0xffffffff  
00000012: PUSH29 0x100000000...  
00000030: PUSH1 0x0  
00000032: CALLDATALOAD  
00000033: DIV  
00000034: AND  
00000035: PUSH4 0x6ae17ab7  
0000003a: DUP2  
0000003b: EQ  
0000003c: PUSH2 0x5b  
0000003f: JUMPI
```

```
----- BLOCK ~40 -----  
00000040: DUP1  
00000041: PUSH4 0x771c0ad9  
00000046: EQ  
00000047: PUSH2 0x8d  
0000004a: JUMPI
```



# Ανάλυση 4/12

- Μετατρέπει τις εντολές σε μορφή SSA

```
----- BLOCK ~0 -----  
0x0: .3 = PHI~0-MEM  
0x0: .0 = #80  
0x2: .1 = #40  
0x4: .2 = MSTORE(.3, .1, .0)  
0x5: .4 = #4  
0x7: .5 = CALLDATASIZE  
0x8: .6 = LT(.5, .4)  
0x9: .7 = #56  
0xc: .8 = JUMPI(.7, .6)
```

```
----- BLOCK ~d -----  
0xd: .0 = #ffffffff  
0x12: .1 = #10000...  
0x30: .2 = #0  
0x32: .3 = CALLDATALOAD(.2)  
0x33: .4 = DIV(.3, .1)  
0x34: .5 = AND(.4, .0)  
0x35: .6 = #6ae17ab7  
0x3b: .7 = EQ(.5, .6)  
0x3c: .8 = #5b  
0x3f: .9 = JUMPI(.8, .7)
```

```
----- BLOCK ~40 -----  
0x40: .0 = PHI~40[-1]  
0x41: .1 = #771c0ad9  
0x46: .2 = EQ(.1, .0)  
0x47: .3 = #8d  
0x4a: .4 = JUMPI(.3, .2)
```

# Ανάλυση 5/12

- Σύνδεση των block, αρχική εκτίμηση CFG

```
----- BLOCK ~0 -----  
0x0: .3 = PHI~0-MEM  
0x0: .0 = #80  
0x2: .1 = #40  
0x4: .2 = MSTORE(.3, .1, .0)  
0x5: .4 = #4  
0x7: .5 = CALLDATASIZE  
0x8: .6 = LT(.5, .4)  
0x9: .7 = #56  
0xc: .8 = JUMPI(.7, .6)
```

NT

```
----- BLOCK ~d -----  
0xd: .0 = #ffffffff  
0x12: .1 = #100000...  
0x30: .2 = #0  
0x32: .3 = CALLDATALOAD(.2)  
0x33: .4 = DIV(.3, .1)  
0x34: .5 = AND(.4, .0)  
0x35: .6 = #6ae17ab7  
0x3b: .7 = EQ(.5, .6)  
0x3c: .8 = #5b  
0x3f: .9 = JUMPI(.8, .7)
```

NT

T

```
----- BLOCK ~40 -----  
0x40: .0 = PHI~40[-1](~d.5)  
0x41: .1 = #771c0ad9  
0x46: .2 = EQ(.1, .0)  
0x47: .3 = #8d  
0x4a: .4 = JUMPI(.3, .2)
```

```
----- BLOCK ~5b -----  
0x5c: .0 = CALLVALUE  
0x5e: .1 = ISZERO(.0)  
0x5f: .2 = #67  
0x62: .3 = JUMPI(.2, .1)
```

# Ανάλυση 6/12

- Κάνει βελτιστοποιήσεις, με τον αλγόριθμο worklist

```
----- BLOCK ~0 -----  
0x0: .3 = PHI~0-MEM  
0x0: .0 = #80  
0x2: .1 = #40  
0x4: .2 = MSTORE(.3, .1, .0)  
0x5: .4 = #4  
0x7: .5 = CALLDATASIZE  
0x8: .6 = LT(.5, .4)  
0x9: .7 = #56  
0xc: .8 = JUMPI(.7, .6)
```

NT

```
----- BLOCK ~d -----  
0xd: .10 = PHI~d-MEM(~0.2)  
0xd: .0 = #ffffffff  
0x12: .1 = #10000...  
0x30: .2 = #0  
0x32: .3 = CALLDATALOAD(.2)  
0x33: .4 = DIV(.3, .1)  
0x34: .5 = AND(.4, .0)  
0x35: .6 = #6ae17ab7  
0x3b: .7 = EQ(.5, .6)  
0x3c: .8 = #5b  
0x3f: .9 = JUMPI(.8, .7)
```

NT

T

```
----- BLOCK ~40 -----  
0x40: .0 = PHI~40[-1](~d.5)  
0x41: .1 = #771c0ad9  
0x46: .2 = EQ(.1, .0)  
0x47: .3 = #8d  
0x4a: .4 = JUMPI(.3, .2)
```

```
----- BLOCK ~5b -----  
0x5b: .4 = PHI~5b-MEM(~d.10)  
0x5c: .0 = CALLVALUE  
0x5e: .1 = ISZERO(.0)  
0x5f: .2 = #67  
0x62: .3 = JUMPI(.2, .1)
```

```
----- BLOCK ~8d -----  
0x8e: .0 = CALLVALUE  
0x90: .1 = ISZERO(.0)  
0x91: .2 = #99  
0x94: .3 = JUMPI(.2, .1)
```

NT

b\_95

```
----- BLOCK ~99 -----  
0x99: .0 = PHI~99[-1]  
0x9b: .1 = #79  
0x9e: .2 = #...  
0xa0: .3 = CALLDATALOAD(.2)  
0xa1: .4 = #24  
0xa3: .5 = CALLDATALOAD(.4)  
0xa4: .6 = #44  
0xa6: .7 = CALLDATALOAD(.6)  
0xa7: .8 = #10f  
0xaa: .9 = JUMP(.8)
```

# Ανάλυση 7/12

- Επιλέγει τις εντολές που θα μπουν στον predictor, πχ SLOAD, CALL και εξαρτήσεις

```
----- * BLOCK ~0 -----  
*0x0: .3 \ PHI~0-MEM  
0x0: .0 = #80  
0x2: .1 = #40  
*0x4: .2 \ MSTORE(.3, .1#40, .0#80)  
0x5: .4 = #4  
0x7: .5 = CALLDATASIZE  
0x8: .6 = LT(.5, .4#4)  
0x9: .7 = #56  
0xc: .8 \ JUMPI(.7#56, .6)
```

NT

```
----- * BLOCK ~d -----  
*0xd: .10 \ PHI~d-MEM(~0.2)  
0xd: .0 = #ffffffff  
0x12: .1 = #10000...  
0x30: .2 = #0  
*0x32: .3 = CALLDATALOAD(.2#0)  
*0x33: .4 = DIV(.3, .1#1000)  
*0x34: .5 = AND(.4, .0#ffff)  
0x35: .6 = #6ae17ab7  
*0x3b: .7 = EQ(.5, .6#6ae1)  
0x3c: .8 = #5b  
*0x3f: .9 \ JUMPI(.8#5b, .7)
```

NT

T

```
----- BLOCK ~40 -----  
0x40: .0 = PHI~40[-1](~d.5)  
0x41: .1 = #771c0ad9  
0x46: .2 = EQ(.1#771c, .0)  
0x47: .3 = #8d  
0x4a: .4 \ JUMPI(.3#8d, .2)
```

```
----- * BLOCK ~5b -----  
*0x5b: .4 \ PHI~5b-MEM(~d.10)  
*0x5c: .0 = CALLVALUE  
*0x5e: .1 = ISZERO(.0)  
0x5f: .2 = #67  
*0x62: .3 \ JUMPI(.2#67, .1)
```

# Ανάλυση 8/12

- Απαριθμεί τις τιμές των επιλεγμένων εντολών

```
----- ON MAP -----
```

```
1 = #40
```

```
2 = #80
```

```
3 = V~0.2-MSTORE(v~0.3-PHIxb232-0B, #40, #80) -xad80-NV
```

```
4 = #10000...
```

```
5 = #0
```

```
6 = #6ae17ab7
```

```
7 = #5b
```

```
8 = #ffffffff
```

```
9 = V~d.3-CALLDATALOAD(#0) -x15b2
```

```
10 = V~d.4-DIV(v~d.3-CALLDATALOADx15b2, #10000...) -x4ea2
```

```
11 = V~d.5-AND(v~d.4-DIVx4ea2, #ffffffff) -x4954
```

```
12 = V~d.7-EQ(v~d.5-ANDx4954, #6ae17ab7) -x30c9
```

```
13 = V~d.9-JUMPI(#5b, v~d.7-EQx30c9) -x2f1e-NV
```

```
14 = #67
```

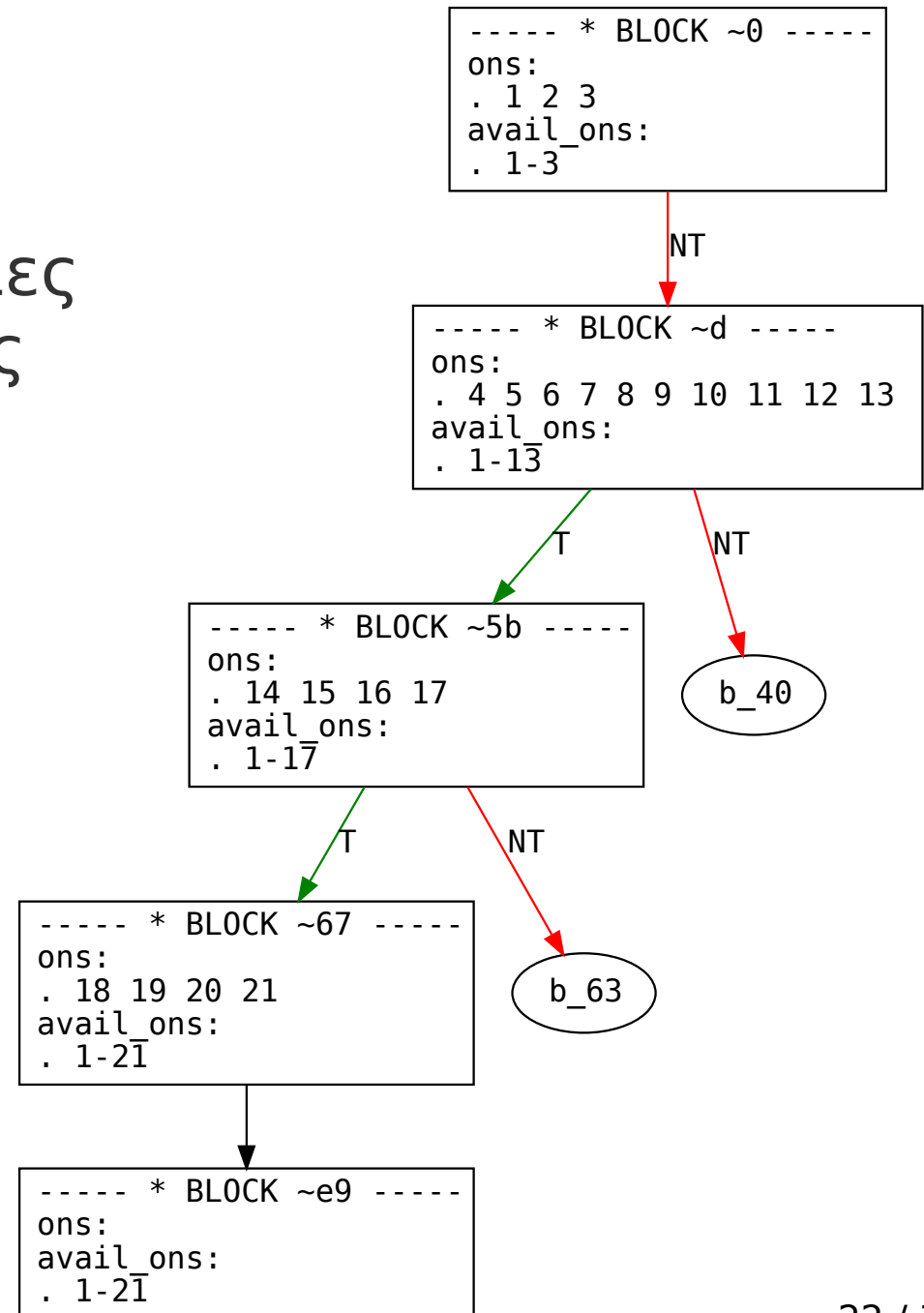
```
15 = V~5b.0-CALLVALUE() -x78d0
```

```
16 = V~5b.1-ISZERO(v~5b.0-CALLVALUEx78d0) -x8a44
```

```
17 = V~5b.3-JUMPI(#67, v~5b.1-ISZEROx8a44) -x9d52-NV
```

# Ανάλυση 9/12

- Βρίσκει σε κάθε block ποιες είναι διαθέσιμες και ποιες πρέπει να υπολογιστούν



# Ανάλυση 10/12

- Υπολογίζει τις νέες εκφράσεις τους

```
----- ON CALCS -----  
0 = ON_0_RESERVED  
1 = #40  
2 = #80  
3 = MSTORE 0 1 2  
4 = #10000...  
5 = #0  
6 = #6ae17ab7  
7 = #5b  
8 = #ffffffff  
9 = CALLDATALOAD 5  
10 = DIV 9 4  
11 = AND 10 8  
12 = EQ 11 6  
13 = JUMPI 7 12  
14 = #67  
15 = CALLVALUE  
16 = ISZERO 15  
17 = JUMPI 14 16  
18 = #24
```

# Ανάλυση 11/12

- Συνθέτει τον κώδικα του predictor

```
~0 | ENTRY
    1 = #40
    2 = #80
    3 = MSTORE 0 1 2
~d | ~0
    4 = #10000...
    5 = #0
    6 = #6ae17ab7
    7 = #5b
    8 = #ffffffff
    9 = CALLDATALOAD 5
   10 = DIV 9 4
   11 = AND 10 8
   12 = EQ 11 6
   13 = JUMPI 7 12
....
```



100

- Έξοδος σε binary encoding, εισάγεται στη DB

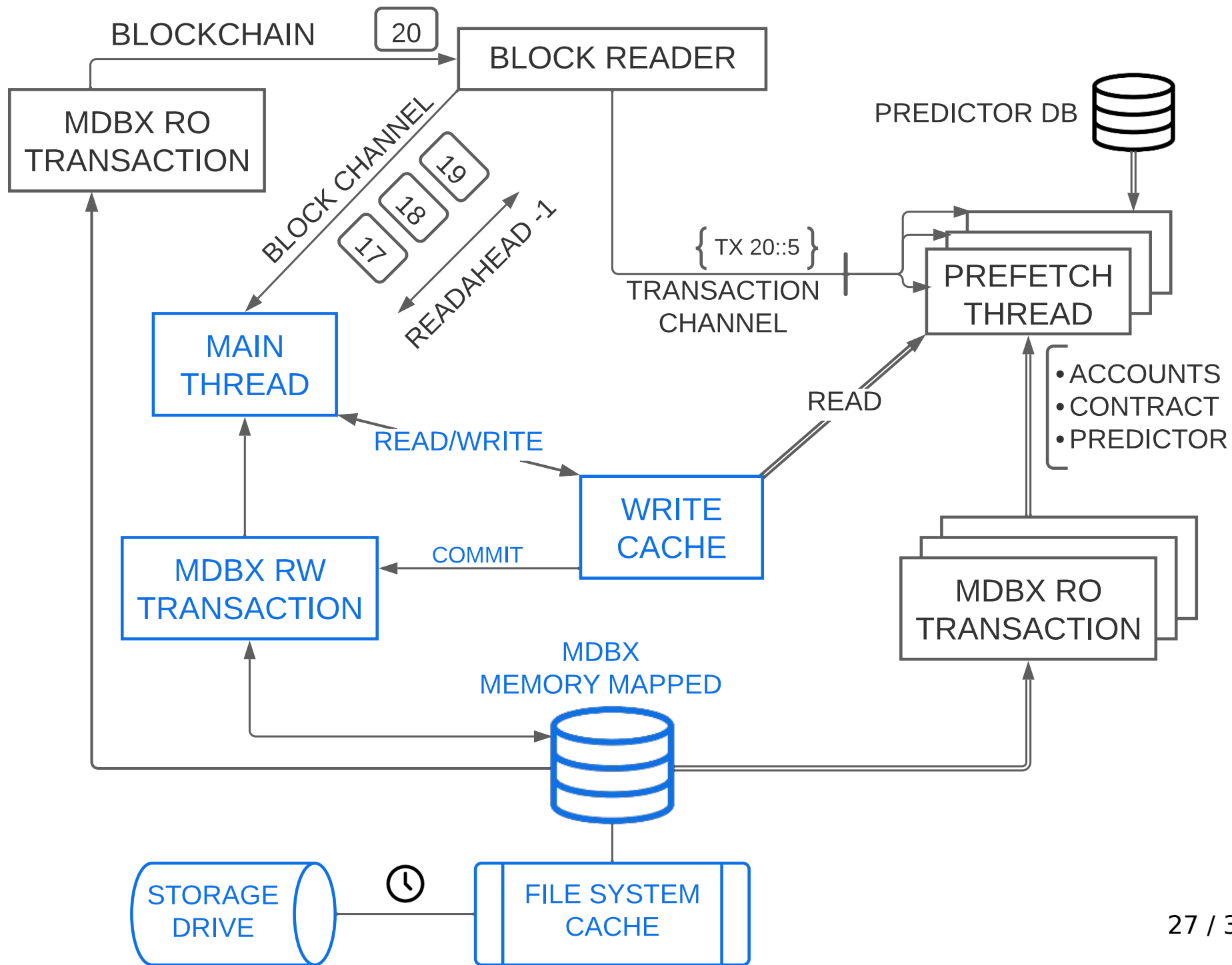
```
Key = 00a5e63813215d7783df9673e42ec7e1d2e5c0896f17e
96ef6f8d28f1e19f663 (original contract's code hash)
```

```
Value = 6a000d001000000100005b00680000010d0067007c0
000015b007900980000010701e900cb0000016700f400d60000
01cd01fc00d9000001f4000501e4000001cd010701ec0000010
5013401f6000002e900fc00a301980100013401b701c3010001
a301cd01e4010001b70101010040010200809001000200220d0
01d0400010000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000
```



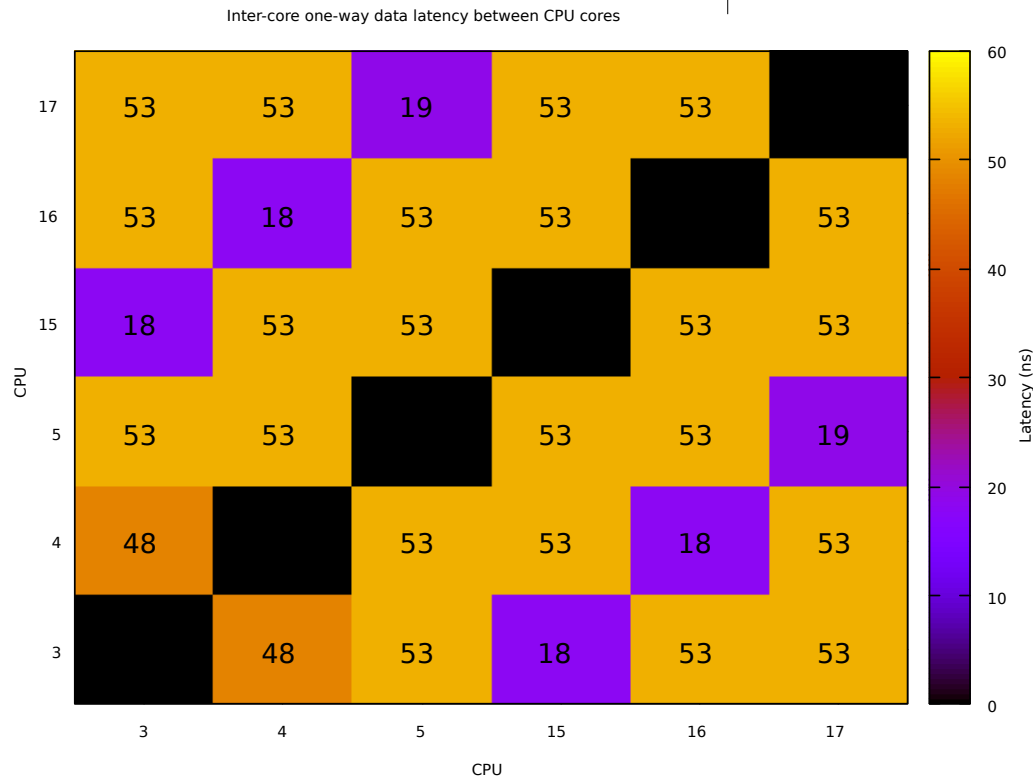
# Prefetching

- Προσθήκη στον erigon, ξεχωριστό go package

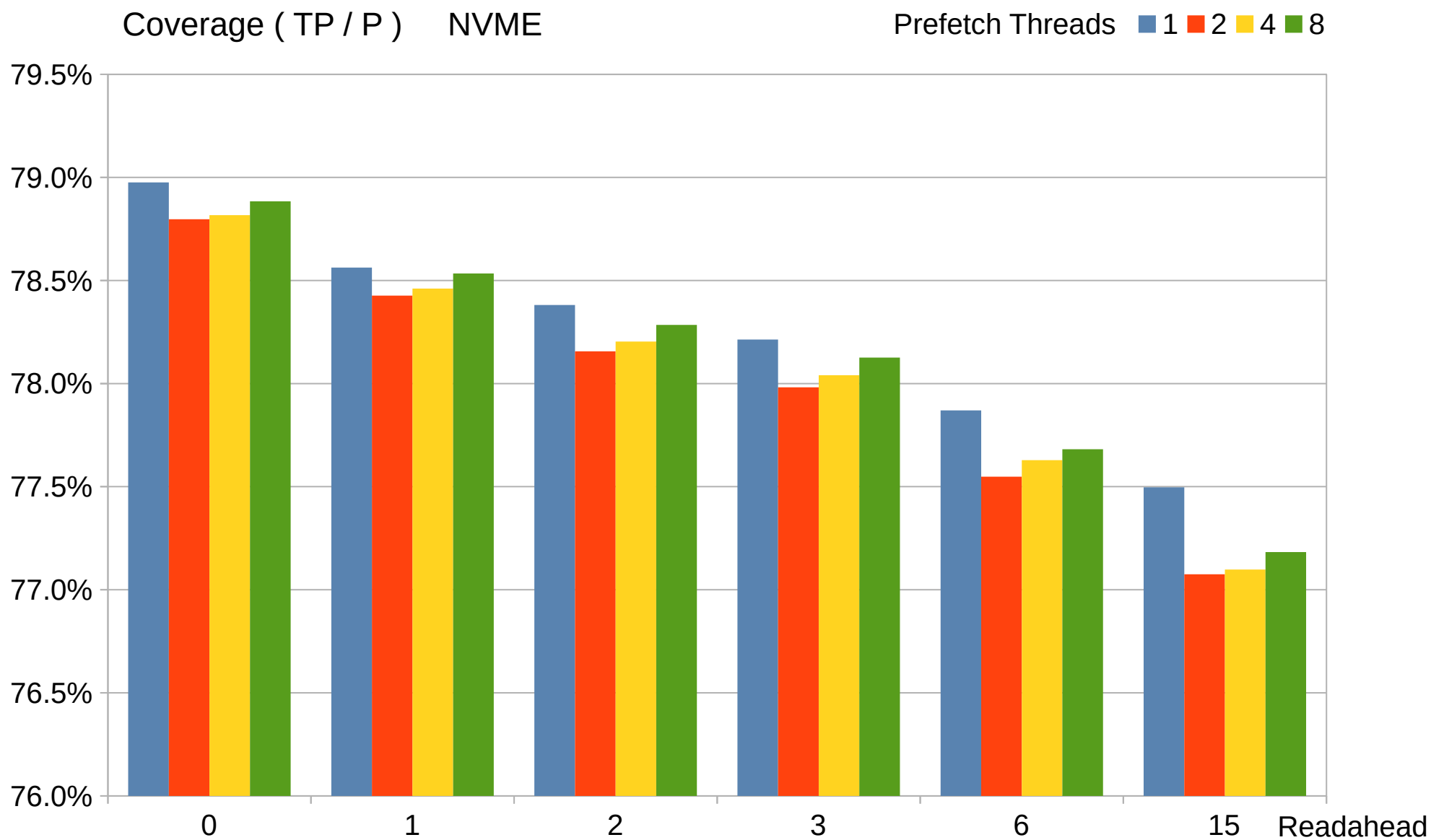


# Πειράματα

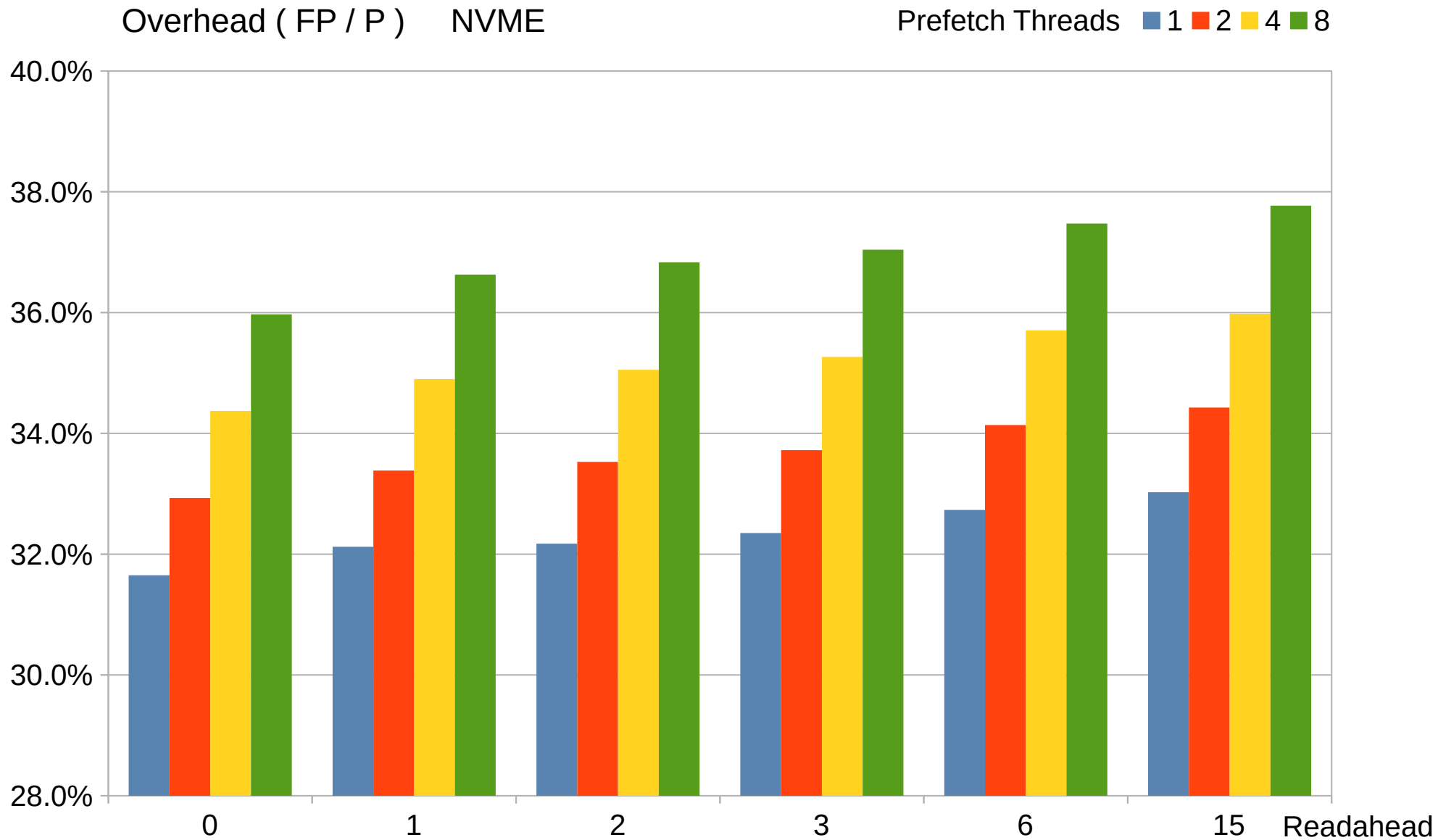
- Όλα στο ίδιο μηχάνημα
- CPU: zen 2, 3.8 GHz, περιορίζουμε σε 1 CCX 3C/6T, κοινή L3
- RAM: περιορίζουμε δεσμεύοντας με ξεχωριστό process
- Δίσκος 1: nvme tlc ssd
- Δίσκος 2: sata qlc ssd



# Readahead και % επιτυχίας

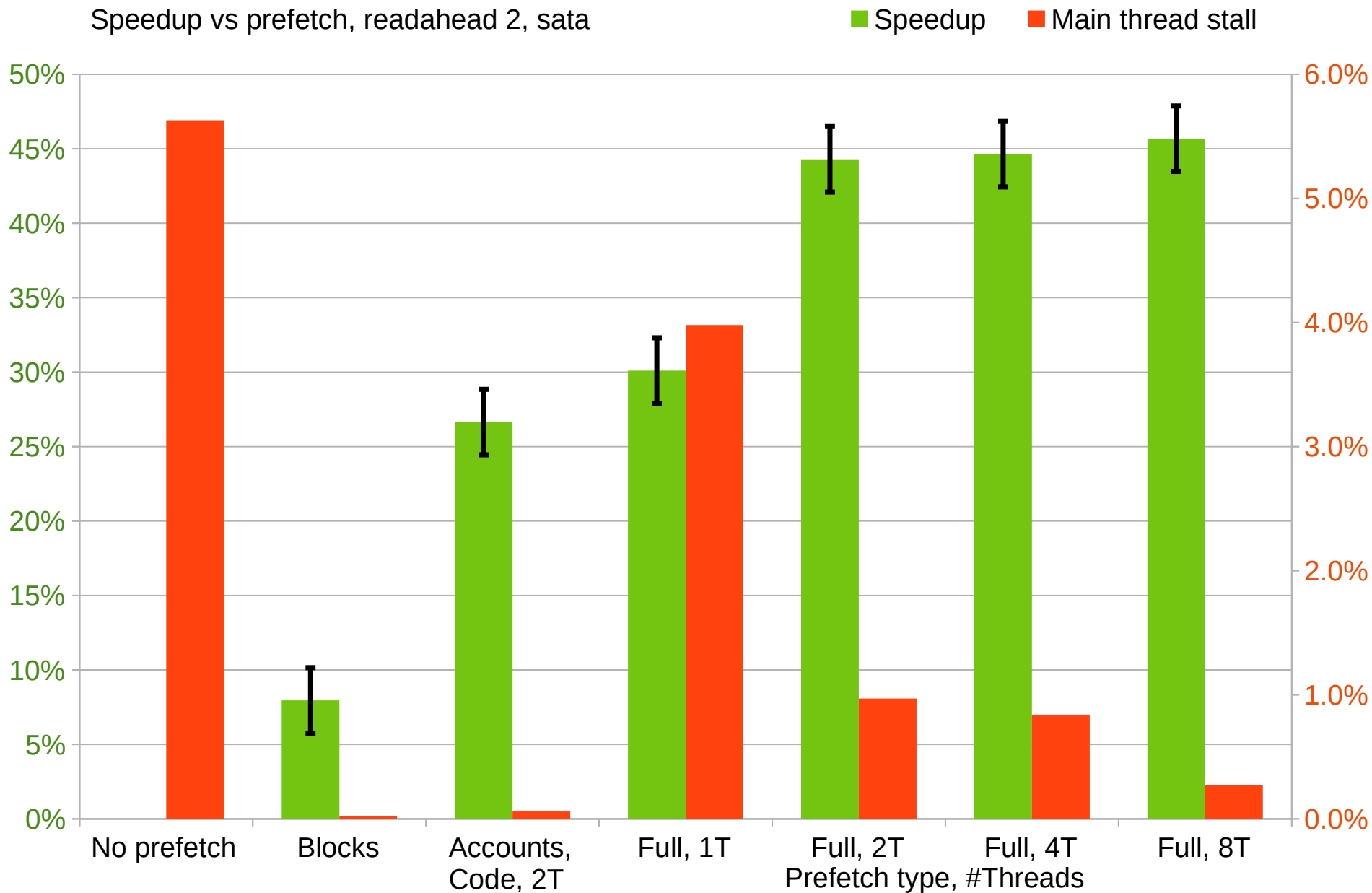


# Readahead και % επιτυχίας

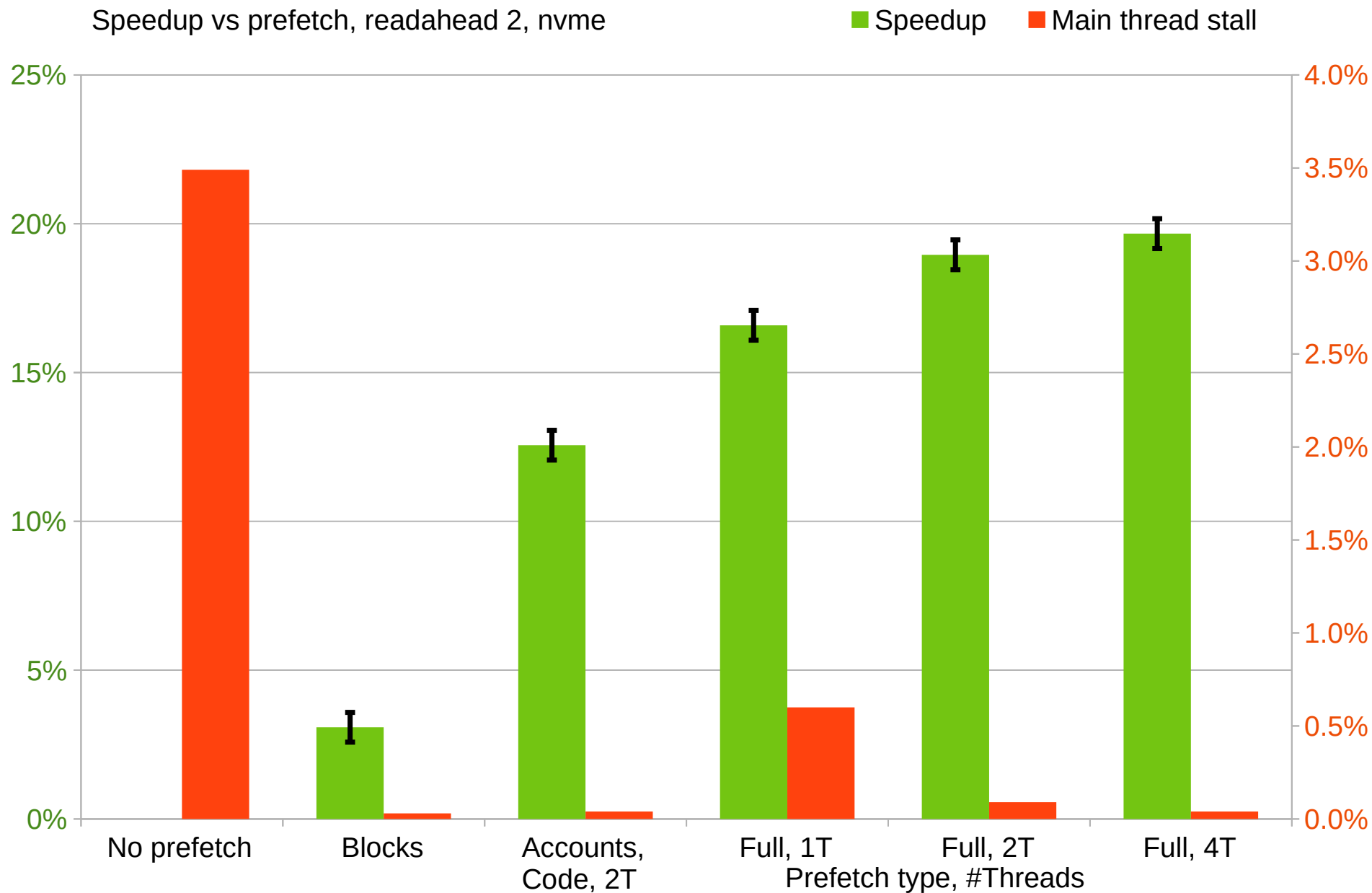


# Speedup, sata

Speedup vs prefetch, readahead 2, sata



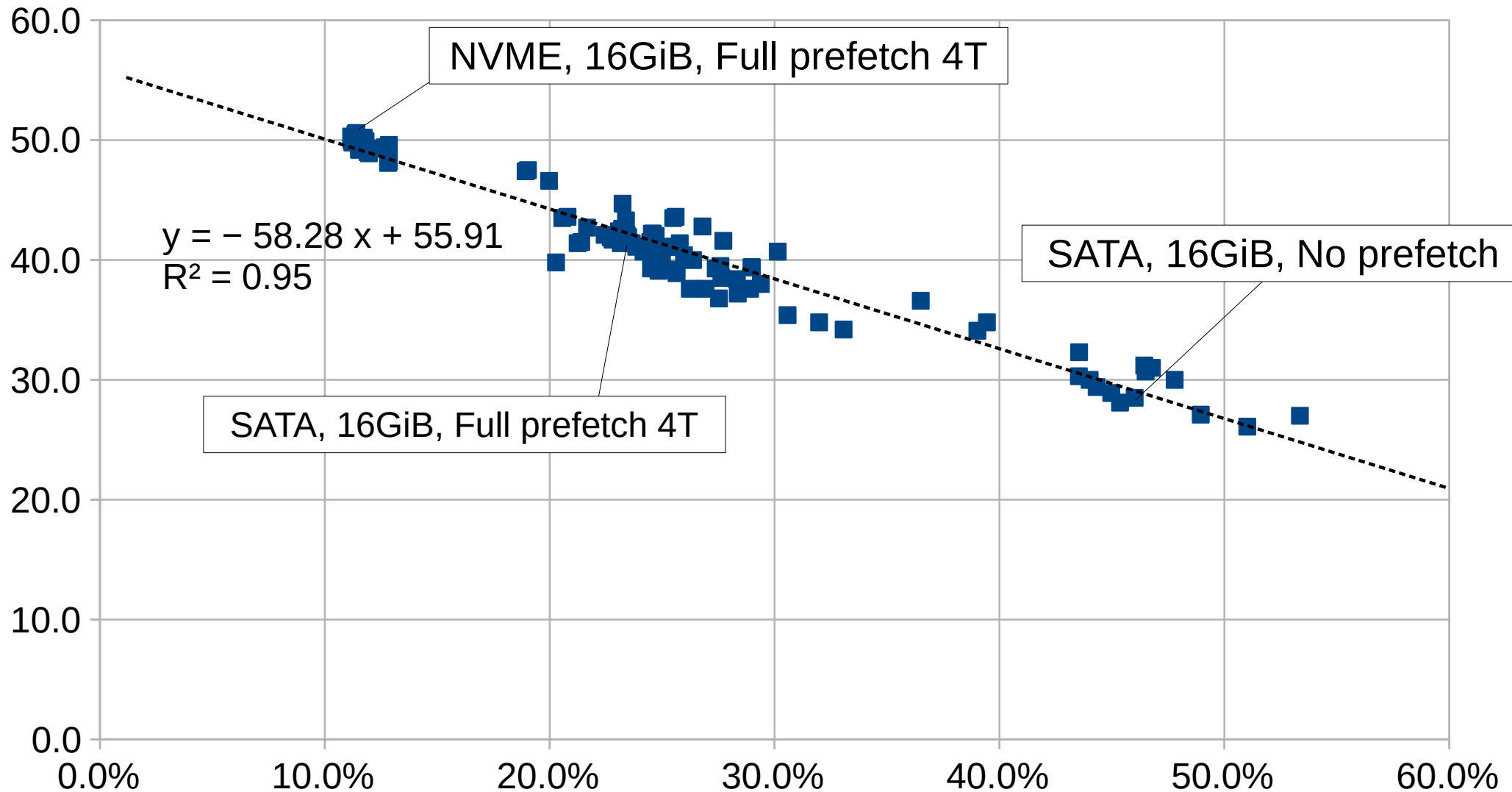
# Speedup, nvme





# Scatter με όλες τις δοκιμές

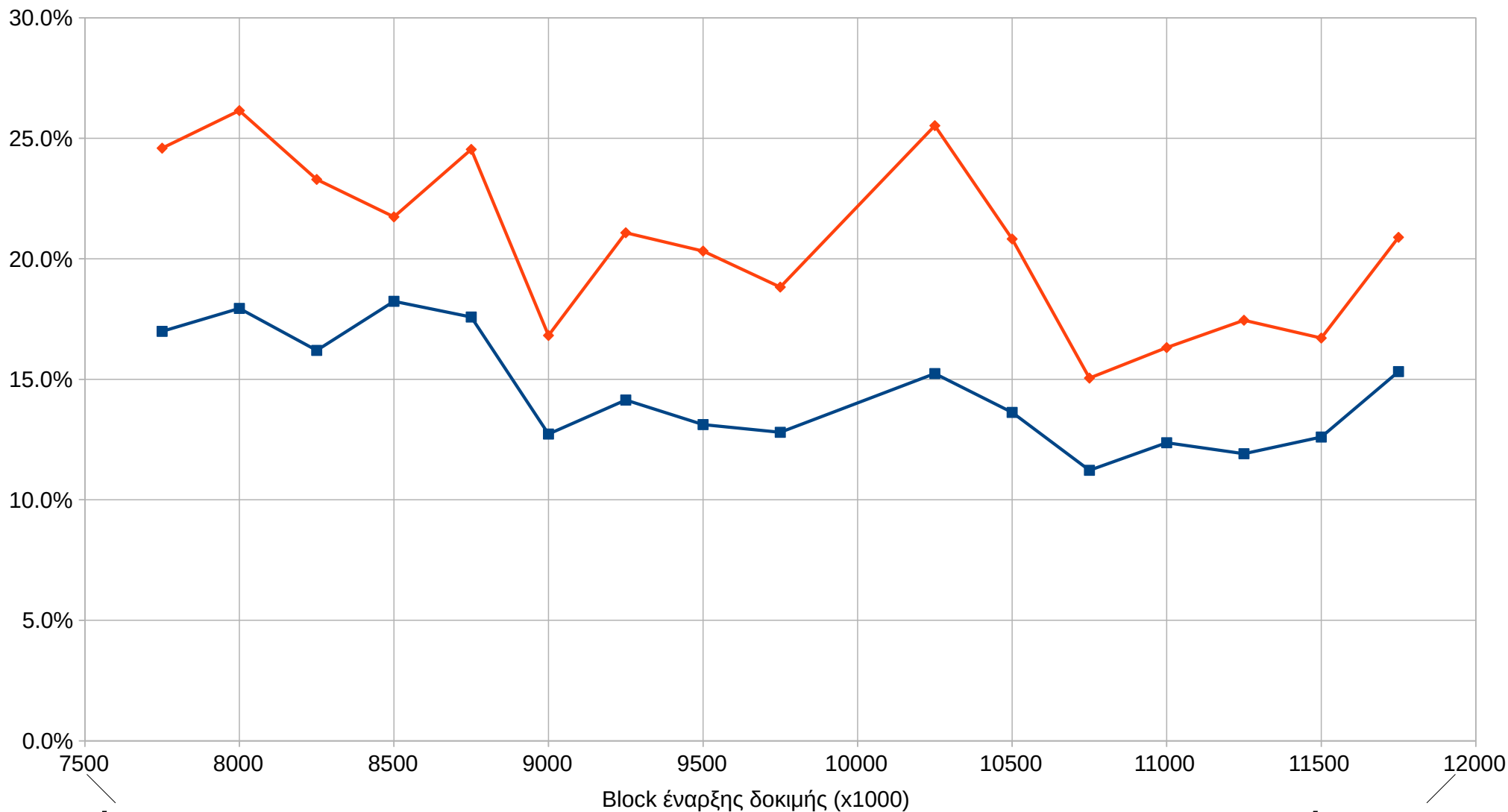
Blocks / second vs % of time main thread waiting DB



# Χρονική εξέλιξη

Speedup σε διαφορετικές χρονικές περιόδους, nvme, 16GiB, 4T

Prefetch type: ■ Accounts & Code ◆ Full



Απρίλιος '19

Μάρτιος '21

# Συμπεράσματα

- Σημαντική βελτίωση σε έναν ήδη βελτιστοποιημένο client
- Αρκετά κοντά στο εκτιμώμενο μέγιστο speedup για τέλεια πρόβλεψη
- Μιας και προβλέπει read/write set των transaction, εκτιμά στην ουσία dependencies τους
  - Δυνατότητα για μελλοντικές επεκτάσεις με παραλληλοποίηση του main execution