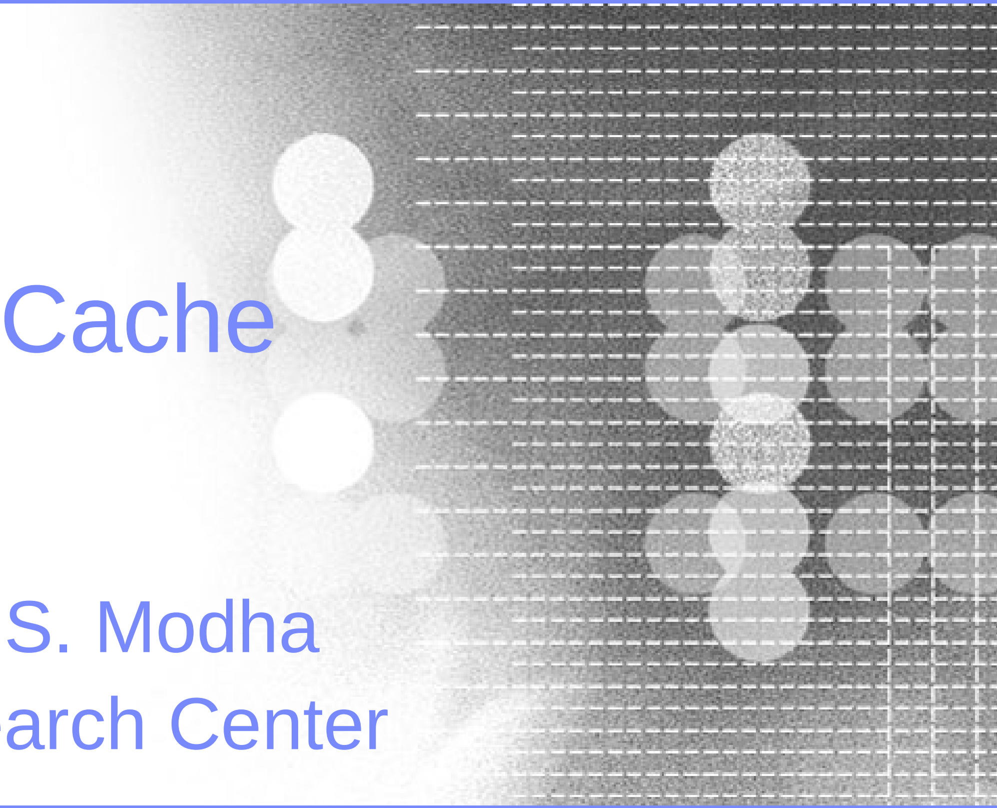
IBM Research



Adaptive Replacement Cache

N. Megiddo and D. S. Modha
IBM Almaden Research Center

Storage Off-site | April 3, 2003 | San Jose, CA

"Computer Science has only three ideas:
cache,

"

Greg Ganger, CMU

"Computer Science has only three ideas:
cache,
hash,
"

Greg Ganger, CMU

"Computer Science has only three ideas:
cache,
hash,
trash."

Greg Ganger, CMU

"...constructing a hierarchy of memories, each of which has greater capacity ... but which is less quickly accessible."

von Neumann et al., 1946

"...constructing a hierarchy of memories,
each of which has greater capacity ...
but which is less quickly accessible."
von Neumann et al., 1946

"Yea, from the table of my memory,
I'll wipe away all trivial fond records"
Shakespeare in Hamlet, 1603

The Replacement Cache Problem

cache: fast but expensive

disks: cheap but slow

How to manage the cache?

Which page to replace?

How to achieve a high hit ratio?

A Brief Survey

- LRU (dates back to 1965, at least)
  - constant time and space complexity & simple-to-implement
  - captures "clustered locality of reference"
  - does not exploit "frequency"
  - is not scan-resistant

A Brief Survey

- LRU (dates back to 1965, at least)
  - constant time and space complexity & simple-to-implement
  - captures "clustered locality of reference"
  - does not exploit "frequency"
  - is not scan-resistant
- LFU (dates back to 1971, at least)
  - exploits "frequency"
  - is scan-resistant
  - logarithmic time complexity (per request)
  - periodic resizing required to prevent stale pages
  - does not capture "clustered locality of reference"

A Brief Survey

- LRU (dates back to 1965, at least)
  - constant time and space complexity & simple-to-implement
  - captures "clustered locality of reference"
  - does not exploit "frequency"
  - is not scan-resistant
- LFU (dates back to 1971, at least)
  - exploits "frequency"
  - is scan-resistant
  - logarithmic time complexity (per request)
  - periodic resizing required to prevent stale pages
  - does not capture "clustered locality of reference"
- LRU + LFU:
  - Log-complexity: LRU-2, LRFU
  - Constant-complexity in expected sense: LIRS, FBR
  - Unbounded Space complexity: LIRS
  - Difficulty of tuning: FBR, LRU-2, 2Q, SLRU, LRFU, LIRS
  - Stringent Assumption on Workload: MQ

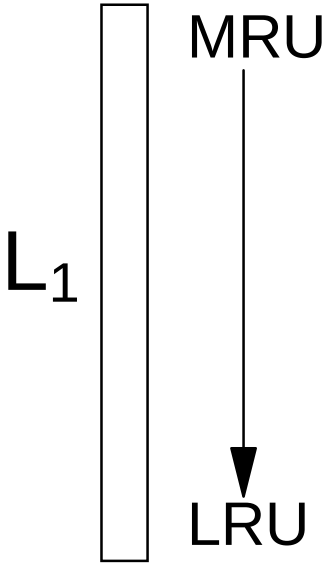
Difficulty of Tuning

- Shasha and Johnson (1994):
  - "difficult to model the tunables of LRU-2"
- Lee et al. (1998):
  - "... parameters in 2Q ... need to be carefully tuned"
- Lee et al. (2002):
  - "... looking for ways to tune the parameters of LRFU"
  - reported LRFU results with best offline choices
- Wong and Wilkes (2002):
  - For SLRU " ... the optimal size ... varied greatly with the workload"

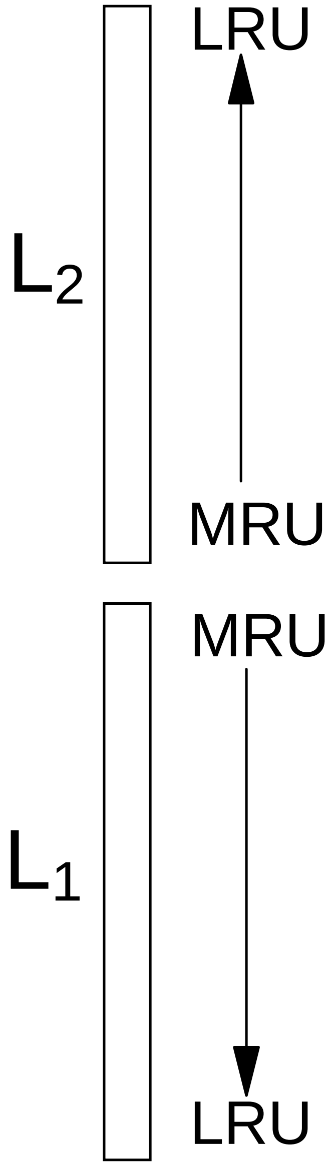
The best tunable parameter values depend upon the workload and the cache size.

Cache Directory (Registry)

- L_1: pages were seen once recently ("recency")

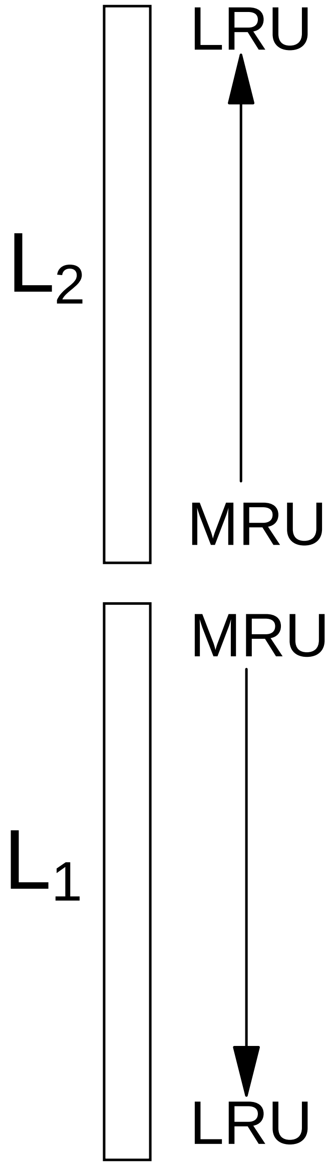


Cache Directory (Registry)



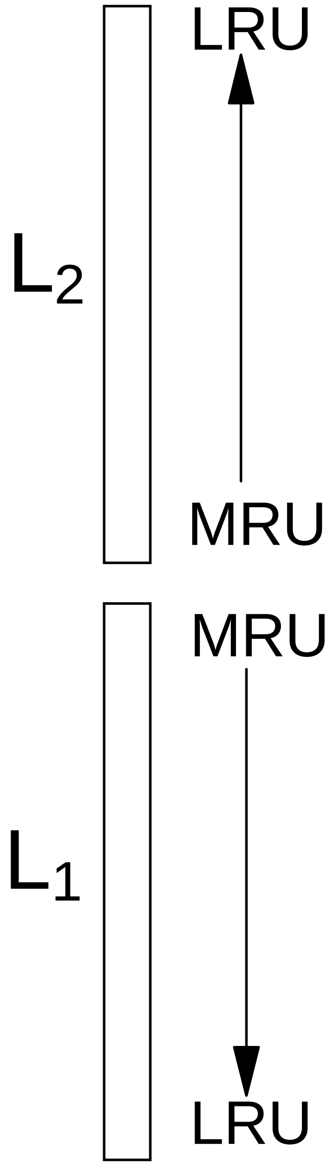
- L_1: pages were seen once recently ("recency")
- L_2: pages were seen at least twice recently ("frequency")

Cache Directory (Registry)



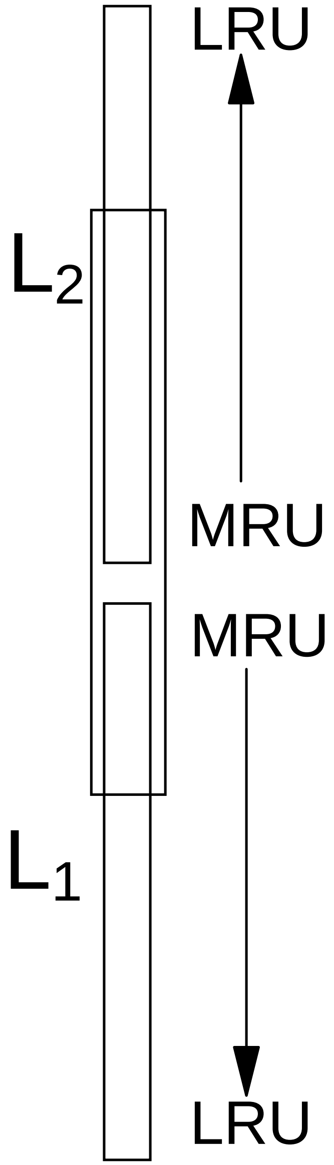
- L_1: pages were seen once recently ("recency")
- L_2: pages were seen at least twice recently ("frequency")
- If L_1 contains exactly c pages
  --replace the LRU page in L_1
  else
  --replace the LRU page in L_2.

Cache Directory (Registry)



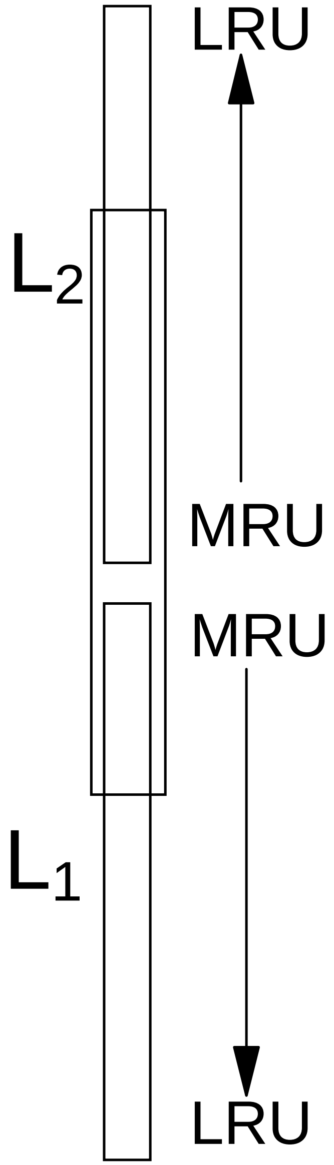
- L_1: pages were seen once recently ("recency")
- L_2: pages were seen at least twice recently ("frequency")
- If L_1 contains exactly c pages
  --replace the LRU page in L_1
  else
  --replace the LRU page in L_2.
- Lemma: The c most recent pages are in the union of L_1 and L_2.

Fixed Replacement Cache (FRC)



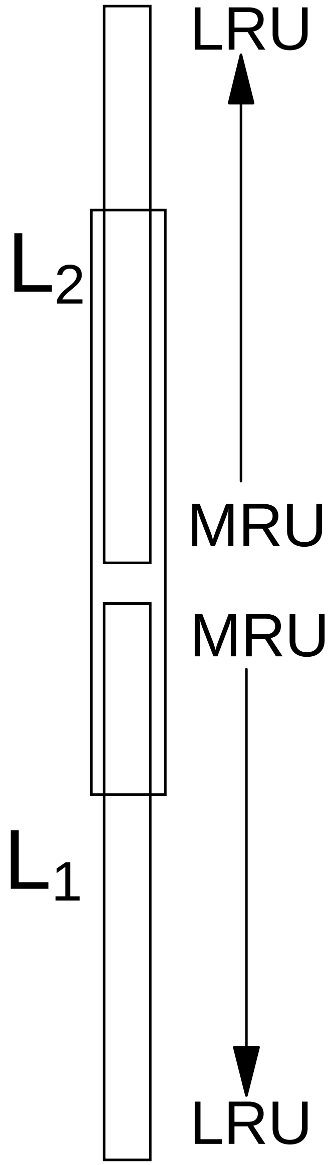
- Divide L_1 into T_1 (top) & B_1 (bottom)

Fixed Replacement Cache (FRC)



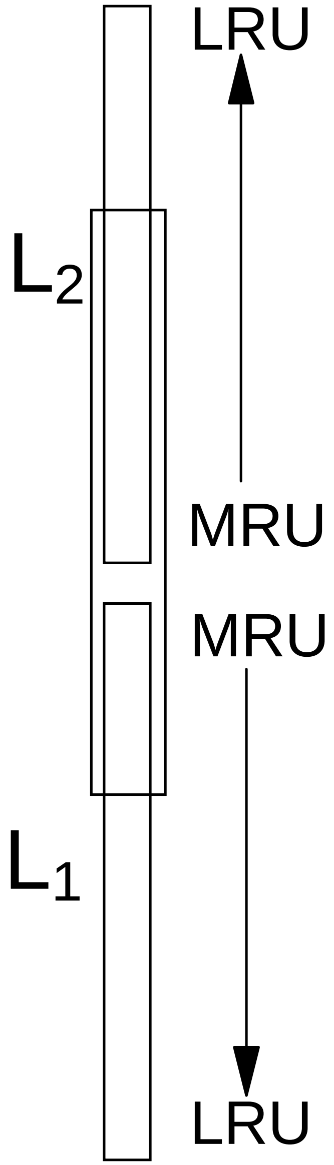
- Divide L_1 into T_1 (top) & B_1 (bottom)
- Divide L_2 into T_2 (top) & B_2 (bottom)

Fixed Replacement Cache (FRC)



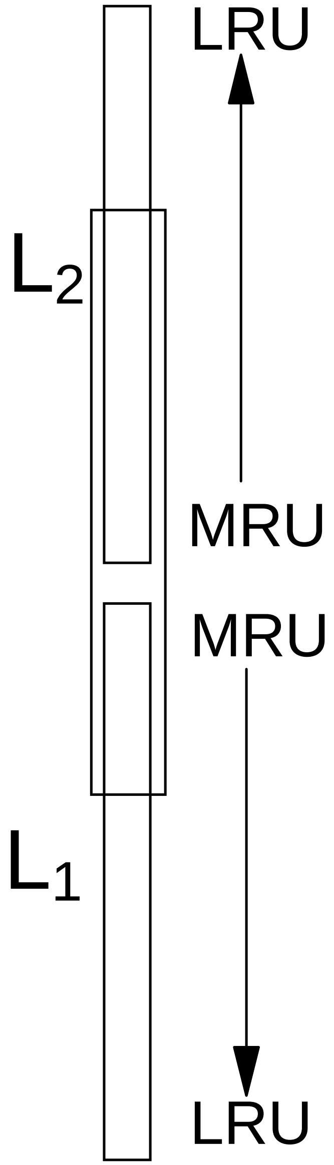
- Divide L_1 into T_1 (top) & B_1 (bottom)
- Divide L_2 into T_2 (top) & B_2 (bottom)
- T_1 and T_2 contain c pages
  - in cache and in directory

Fixed Replacement Cache (FRC)



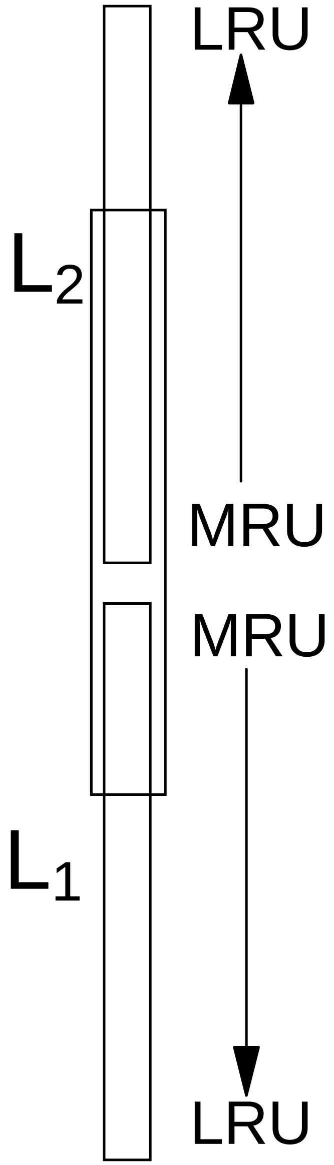
- Divide L_1 into T_1 (top) & B_1 (bottom)
- Divide L_2 into T_2 (top) & B_2 (bottom)
- T_1 and T_2 contain c pages
  - in cache and in directory
- B_1 and B_2 contain c pages
  - in directory, but not in cache

Fixed Replacement Cache (FRC)



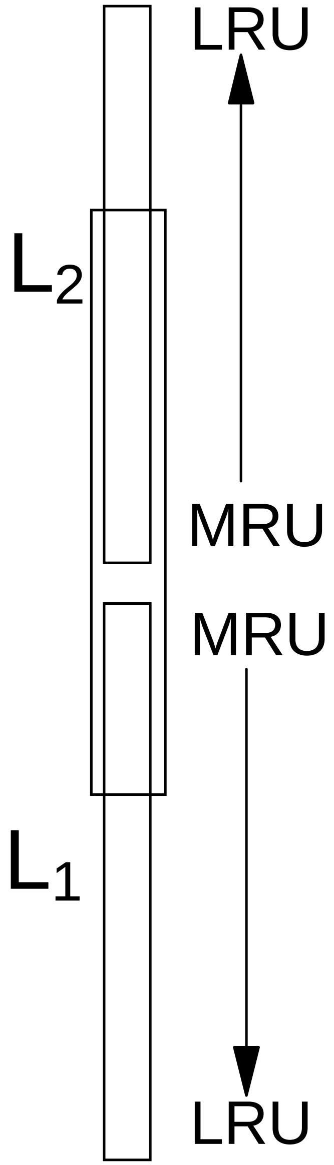
- Divide L_1 into T_1 (top) & B_1 (bottom)
- Divide L_2 into T_2 (top) & B_2 (bottom)
- T_1 and T_2 contain c pages
  - in cache and in directory
- B_1 and B_2 contain c pages
  - in directory, but not in cache
- FRC(p):
  - Set target size of T_1 to p

Fixed Replacement Cache (FRC)



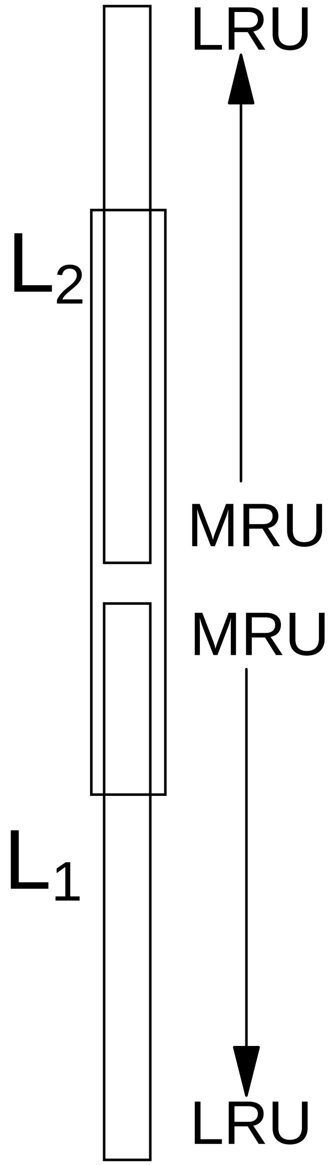
- Divide L_1 into T_1 (top) & B_1 (bottom)
- Divide L_2 into T_2 (top) & B_2 (bottom)
- T_1 and T_2 contain c pages
  - in cache and in directory
- B_1 and B_2 contain c pages
  - in directory, but not in cache
- FRC(p):
  - Set target size of T_1 to p
- If T_1 contains more than p pages,
  --replace LRU page in T_1,
  else
  --replace LRU page in T_2.

Adaptive Replacement Cache (ARC)



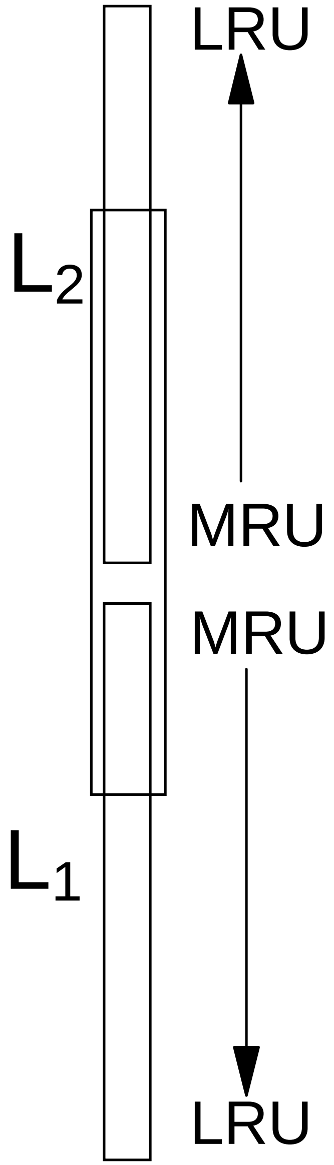
- Adapt target size of T_1 to an observed workload

Adaptive Replacement Cache (ARC)



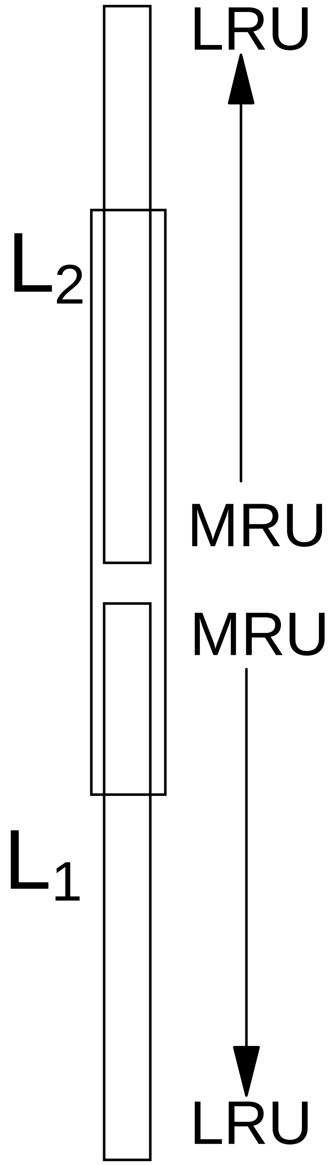
- Adapt target size of T_1 to an observed workload
- A self-tuning algorithm:
  - hit in T_1 or T_2 : do nothing

Adaptive Replacement Cache (ARC)



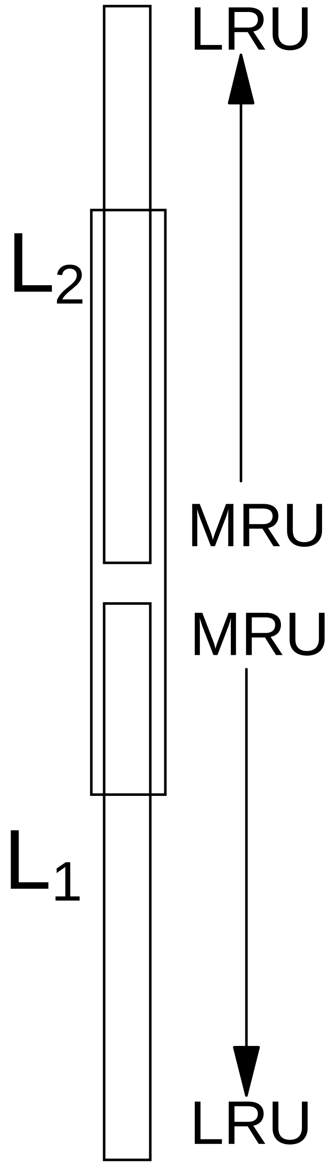
- Adapt target size of T_1 to an observed workload
- A self-tuning algorithm:
  - hit in T_1 or T_2 : do nothing
  - hit in B_1: increase target of T_1

Adaptive Replacement Cache (ARC)

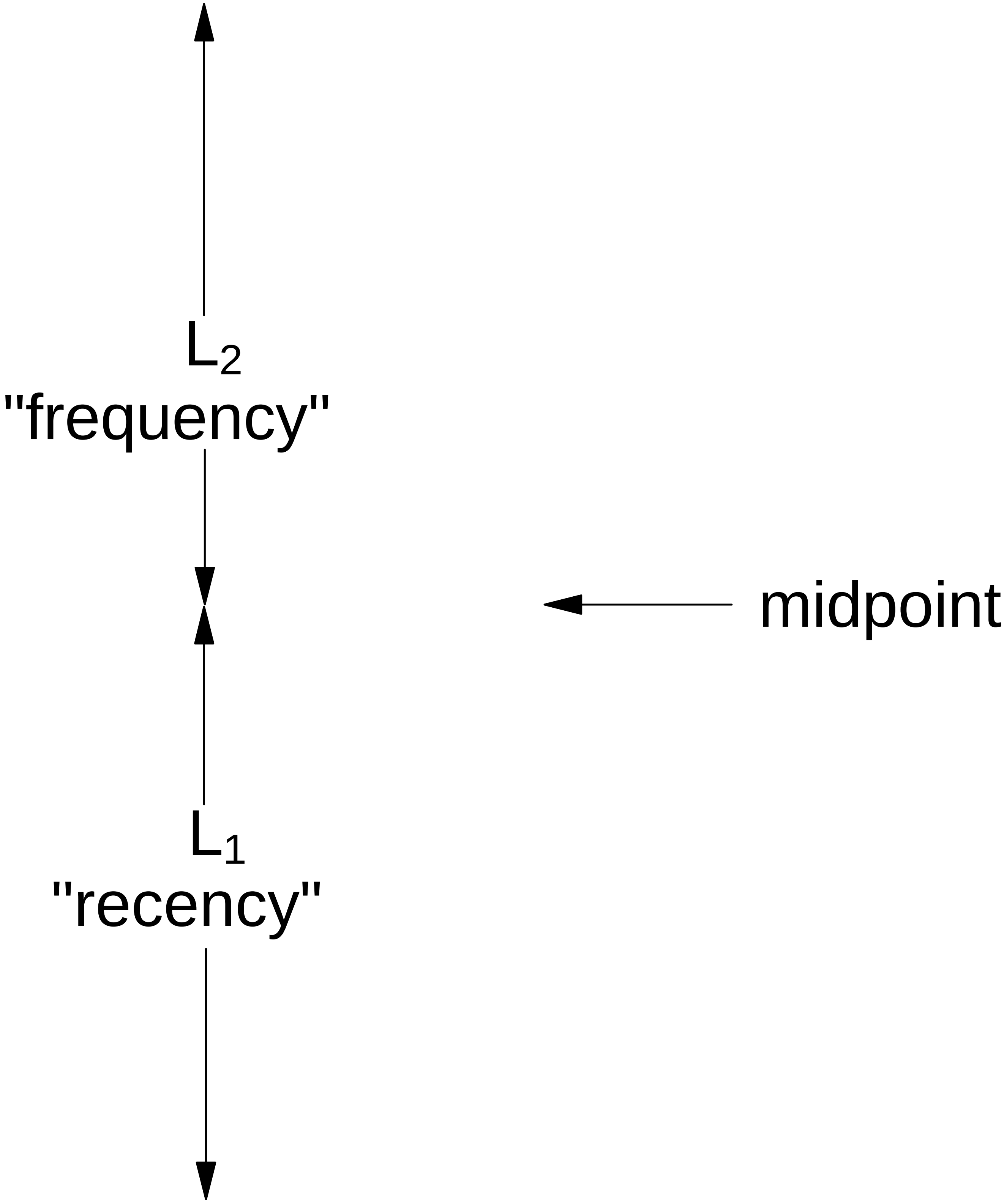


- Adapt target size of T_1 to an observed workload
- A self-tuning algorithm:
  - hit in T_1 or T_2 : do nothing
  - hit in B_1: increase target of T_1
  - hit in B_2: decrease target of T_1

Adaptive Replacement Cache (ARC)



- Adapt target size of T_1 to an observed workload
- A self-tuning algorithm:
  - hit in T_1 or T_2 : do nothing
  - hit in B_1: increase target of T_1
  - hit in B_2: decrease target of T_1
- ARC is scan-resistant


L_2
"frequency"
midpoint
L_1
"recency"

ARC has Low Computational Overhead

Cache Size	LRU	ARC	2Q	LRU-2	LRFU	LRFU	LRFU
					10E-7	10E-3	.99
1024	17	14	17	33	554	408	28
2048	12	14	17	27	599	451	28
4096	12	15	17	27	649	494	29
8192	12	16	18	28	694	537	29
16384	13	16	19	30	734	418	30
32768	14	17	18	31	716	420	31
65536	14	16	18	32	648	424	34
131072	14	15	16	32	533	432	39
262144	13	13	14	30	427	435	42
524288	12	13	13	27	263	443	45

Trace P9

ARC Compares well to LRU, 2Q, LRU-2, LRFU, LIRS

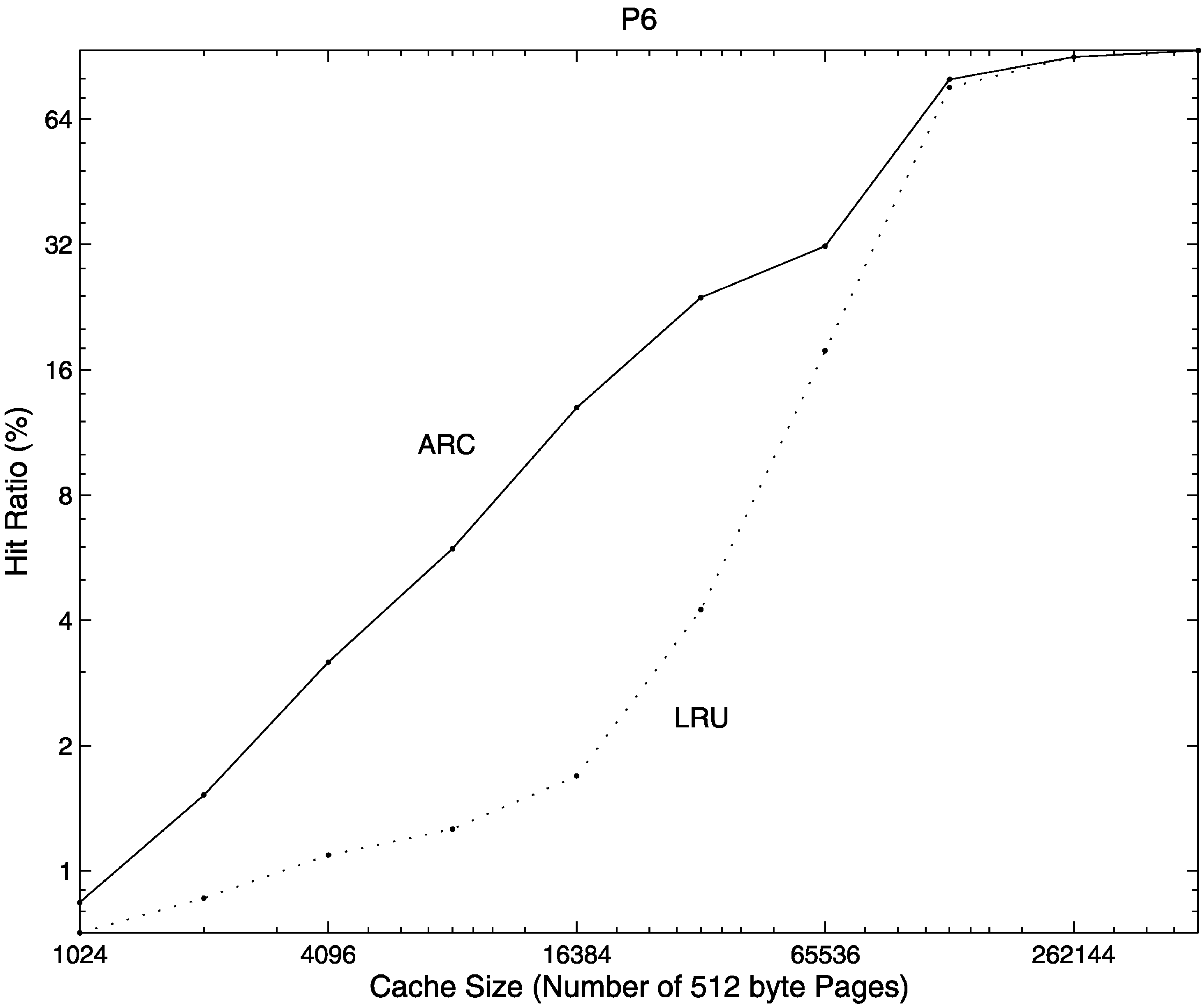
Cache Size	LRU	ARC	2Q	LRU-2	LRFU	LIRS
	ONLINE	ONLINE	OFFLINE	OFFLINE	OFFLINE	OFFLINE
1024	4.09	4.16	4.13	4.07	4.09	4.08
2048	4.84	4.89	4.89	4.83	4.84	4.83
4096	5.61	5.76	5.76	5.81	5.61	5.61
8192	6.22	7.14	7.52	7.54	7.29	6.61
16384	7.09	10.12	11.05	10.67	11.01	9.29
32768	8.93	15.94	16.89	16.36	16.35	15.15
65536	14.43	26.09	27.46	25.79	25.35	25.65
131072	29.21	38.68	41.09	39.58	39.78	40.37
262144	49.11	53.47	53.31	53.43	54.56	53.65
524288	60.91	63.56	61.64	63.15	63.13	63.89

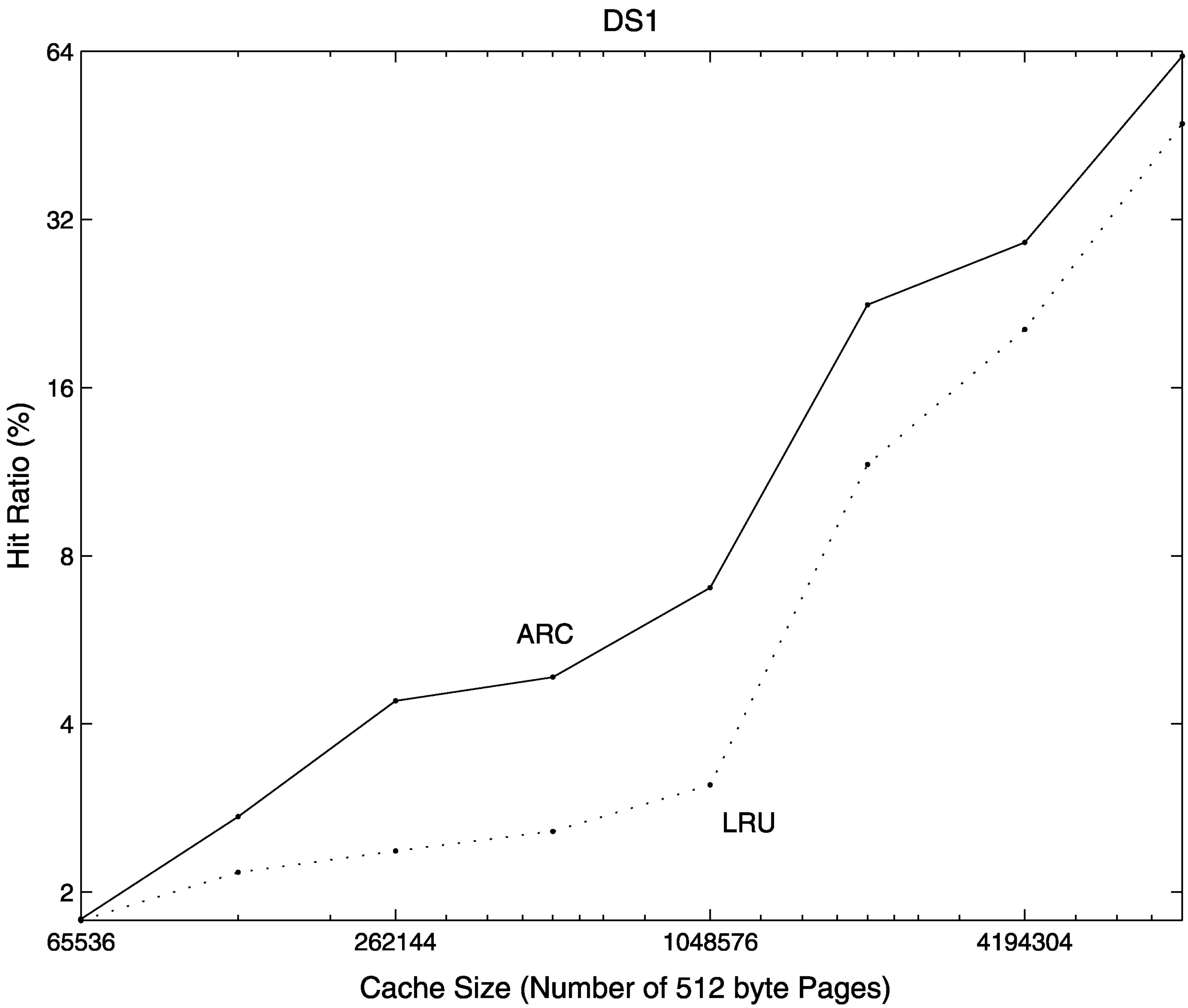
Trace P12

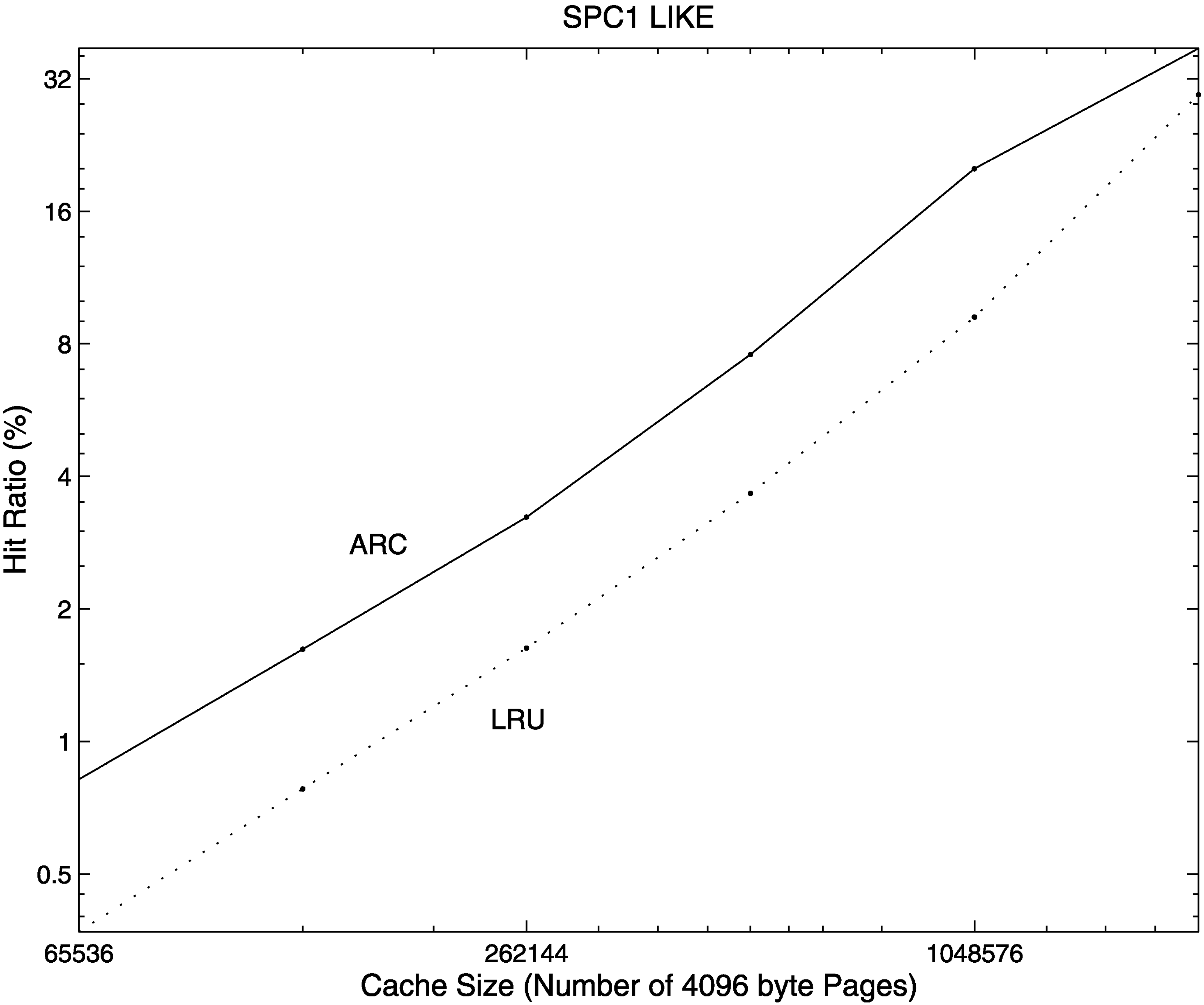
ARC Compares well to LRU, MQ, and 2Q (all online)

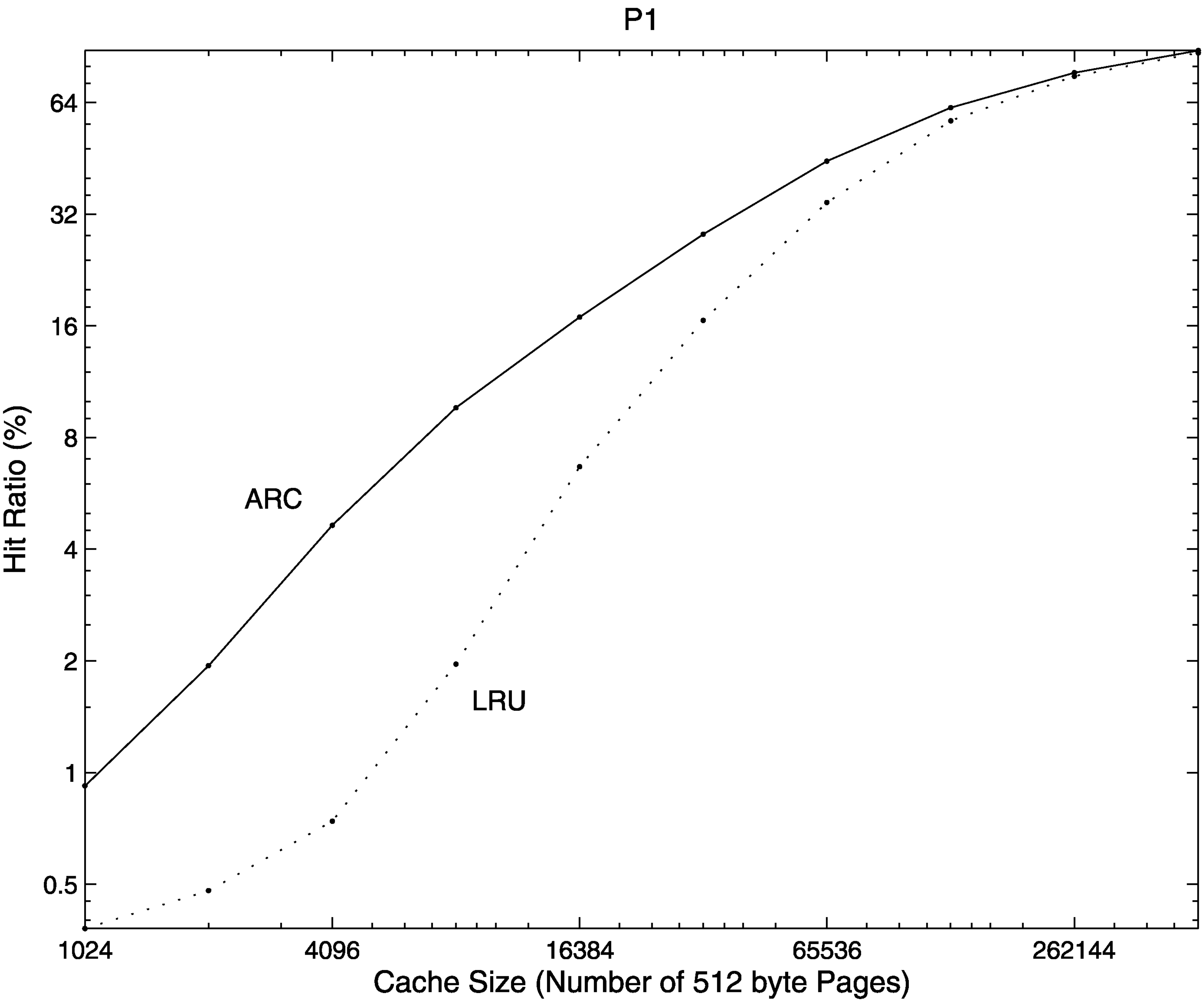
Cache Size	LRU	MQ	2Q	ARC
	ONLINE	ONLINE	ONLINE	ONLINE
16384	0.20	0.20	0.73	1.04
32768	0.40	0.40	1.47	2.08
65536	0.79	0.79	2.85	4.07
131072	1.59	1.59	5.52	7.78
262144	3.23	4.04	10.36	14.30
524288	8.06	14.39	18.89	24.34
1048576	27.62	40.13	35.39	40.44
1572864	50.86	56.49	53.19	57.19
2097152	68.68	70.45	67.36	71.41
4194304	87.30	87.29	86.22	87.26

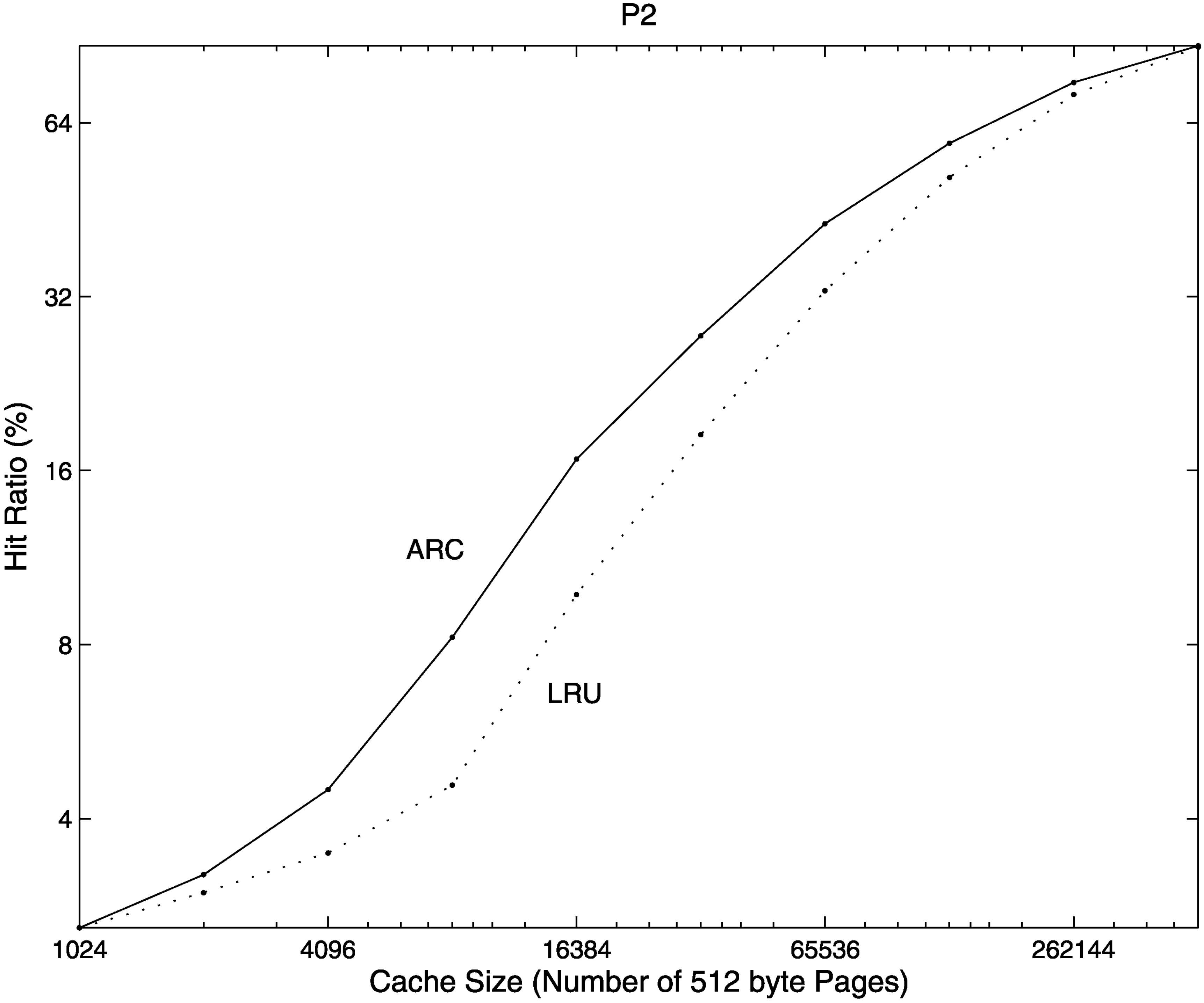
Trace MergeS

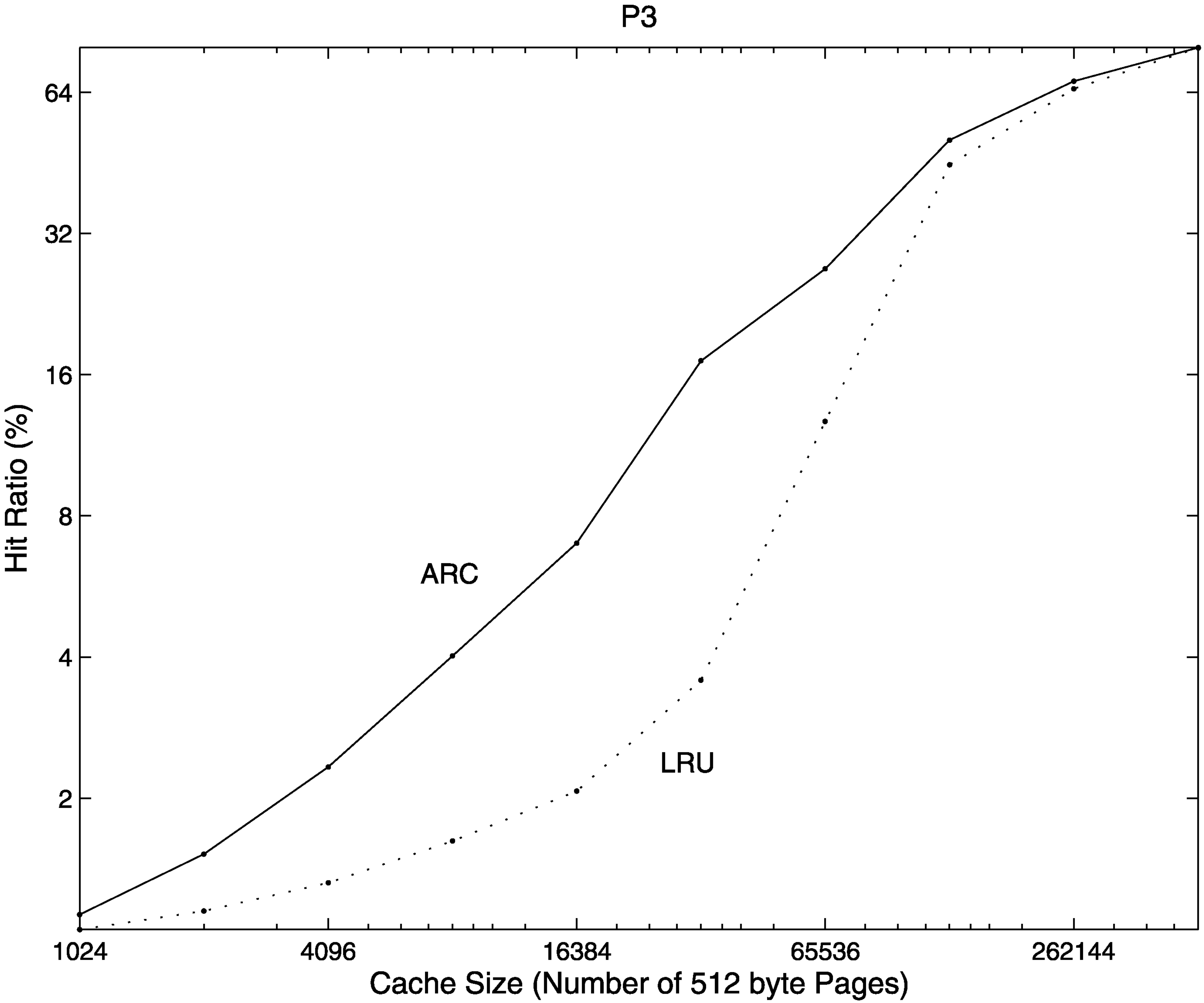

P6
Hit Ratio (%)
64
32
16
8
4
2
1
1024
4096
16384
65536
262144
Cache Size (Number of 512 byte Pages)
ARC
LRU

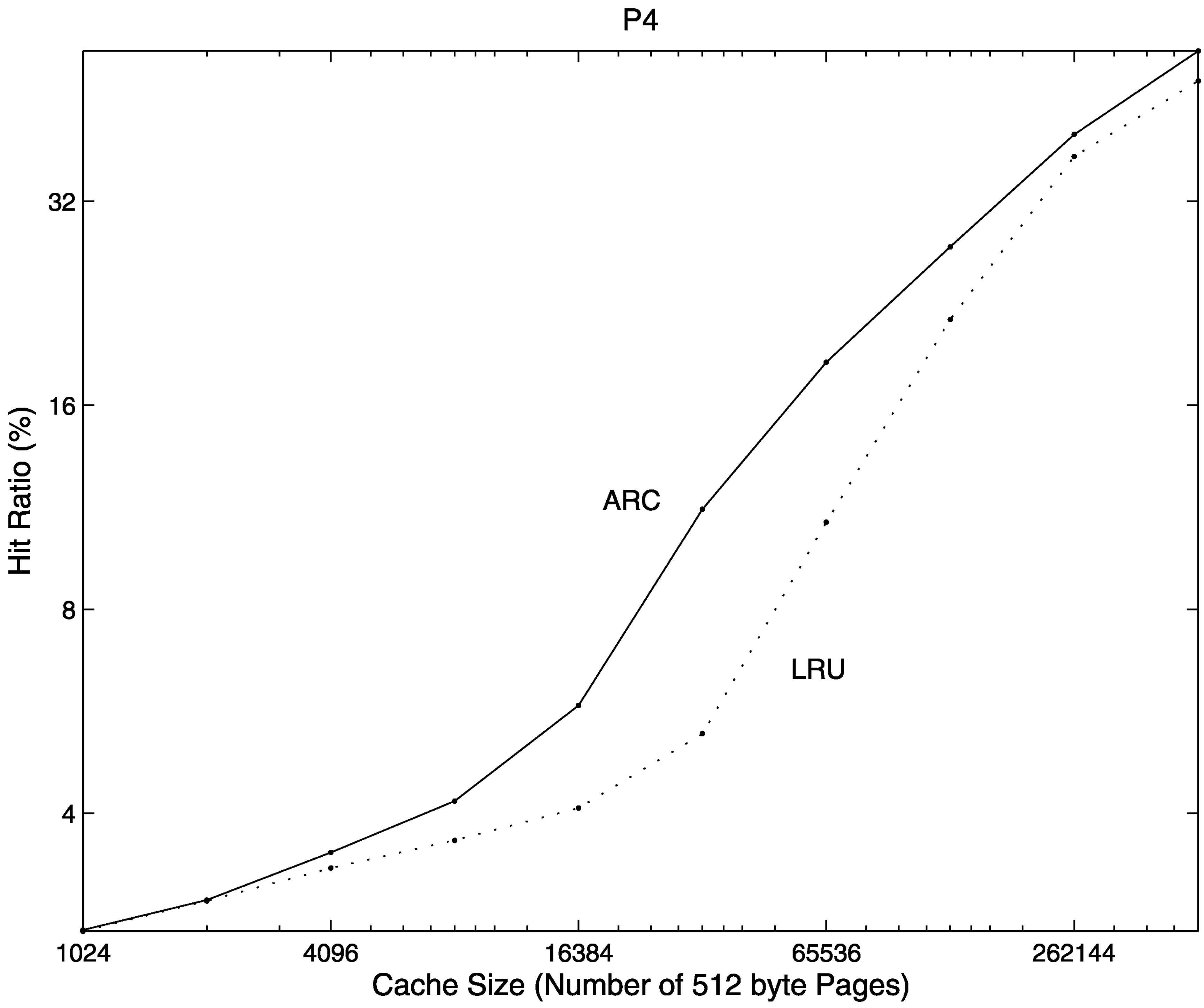

DS1
Hit Ratio (%)
64
32
16
8
4
2
ARC
LRU
65536
262144
1048576
4194304
Cache Size (Number of 512 byte Pages)

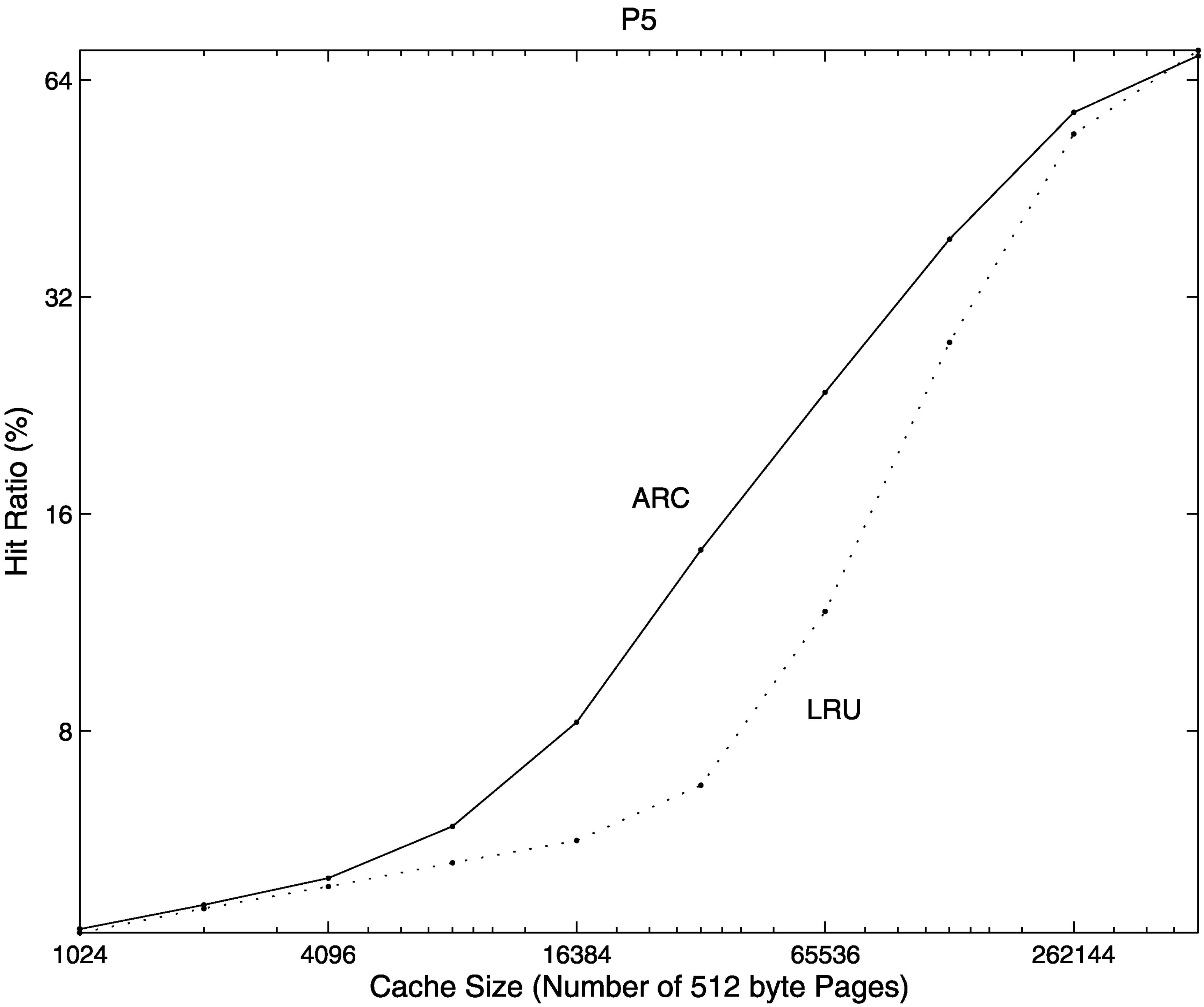

SPC1 LIKE
Hit Ratio (%)
32
16
8
4
2
1
0.5
ARC
LRU
65536
262144
1048576
Cache Size (Number of 4096 byte Pages)

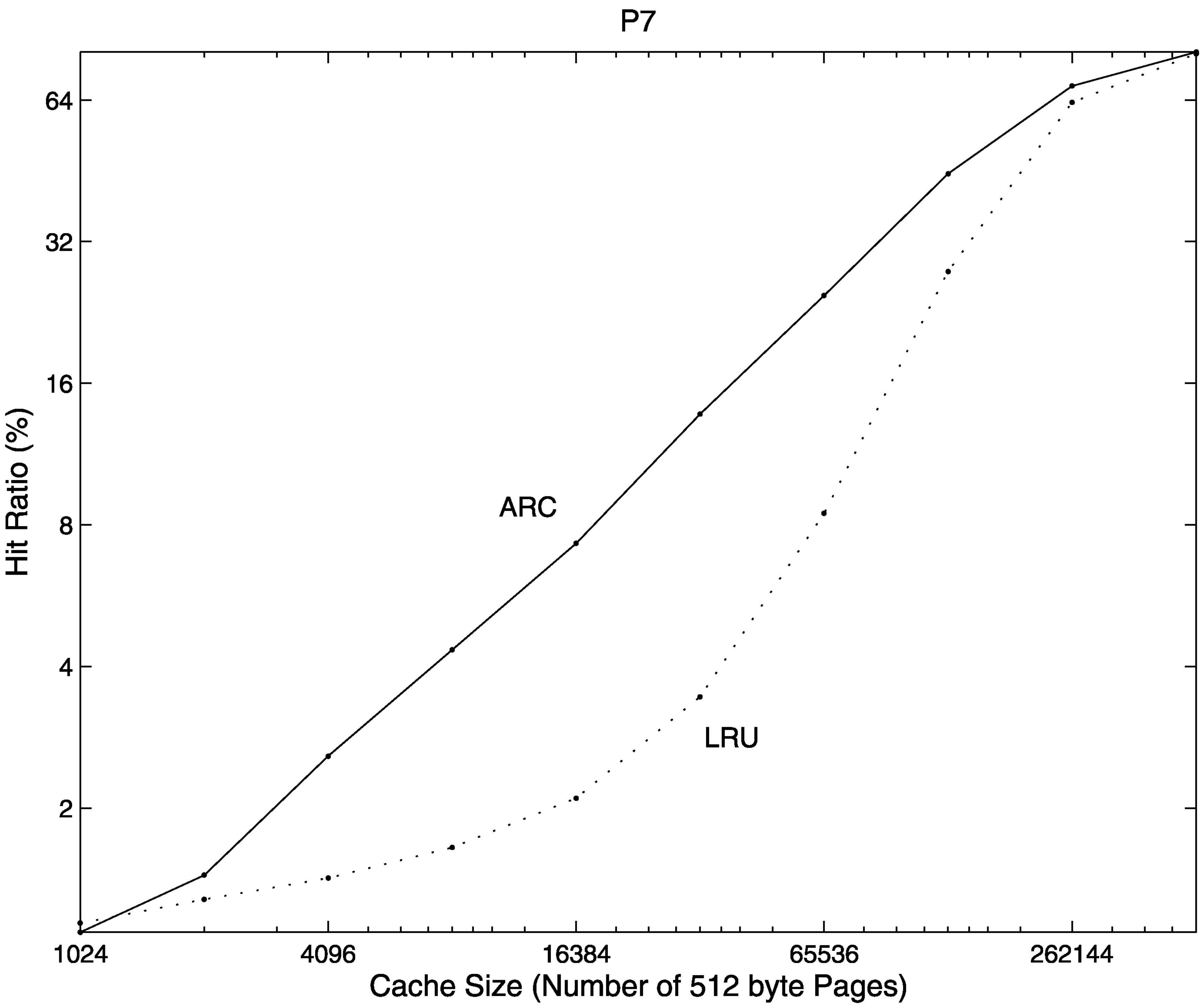

P1
Hit Ratio (%)
64
32
16
8
4
2
1
0.5
ARC
LRU
1024
4096
16384
65536
262144
Cache Size (Number of 512 byte Pages)

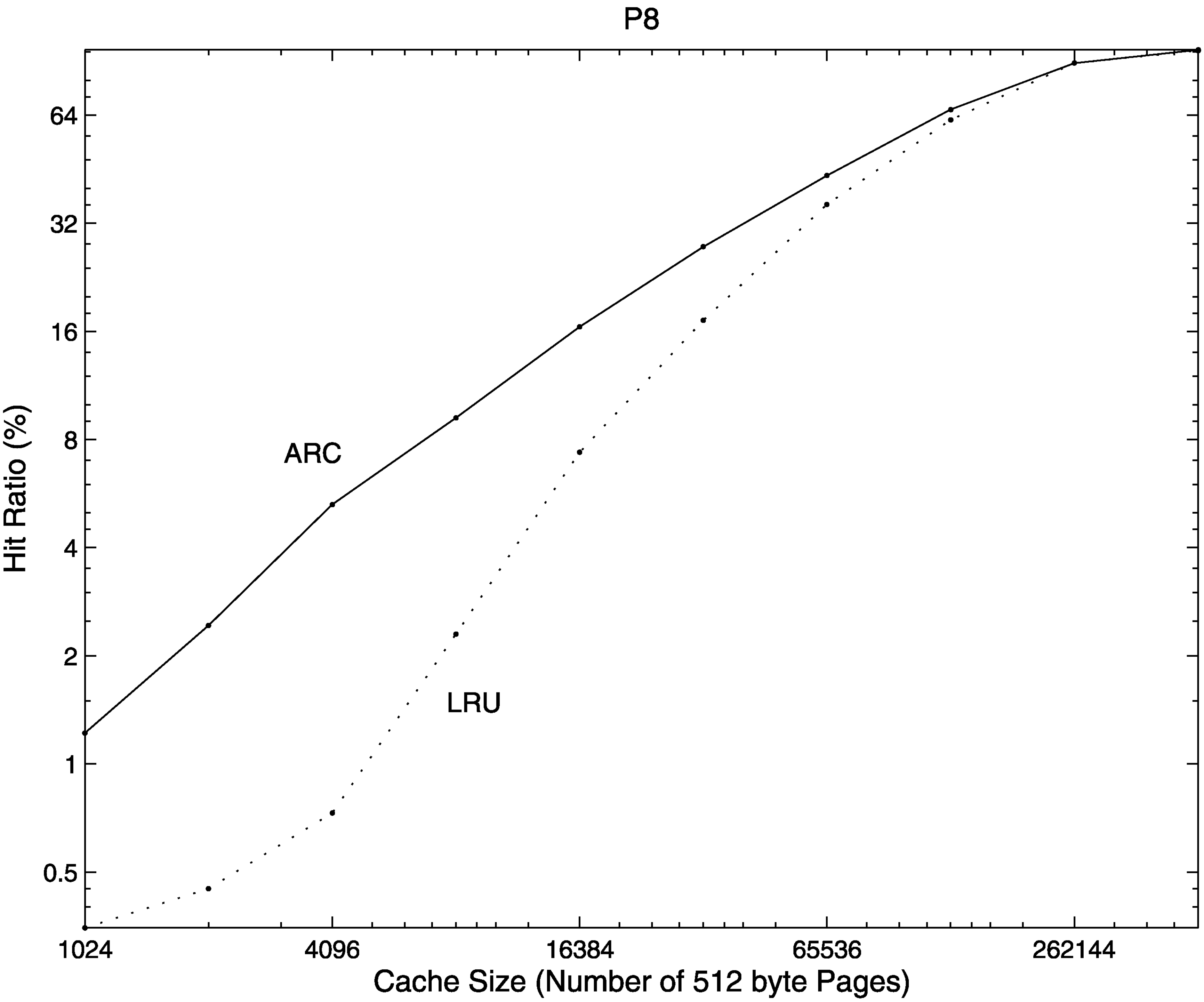

P2
Hit Ratio (%)
64
32
16
8
4
ARC
LRU
1024
4096
16384
65536
262144
Cache Size (Number of 512 byte Pages)

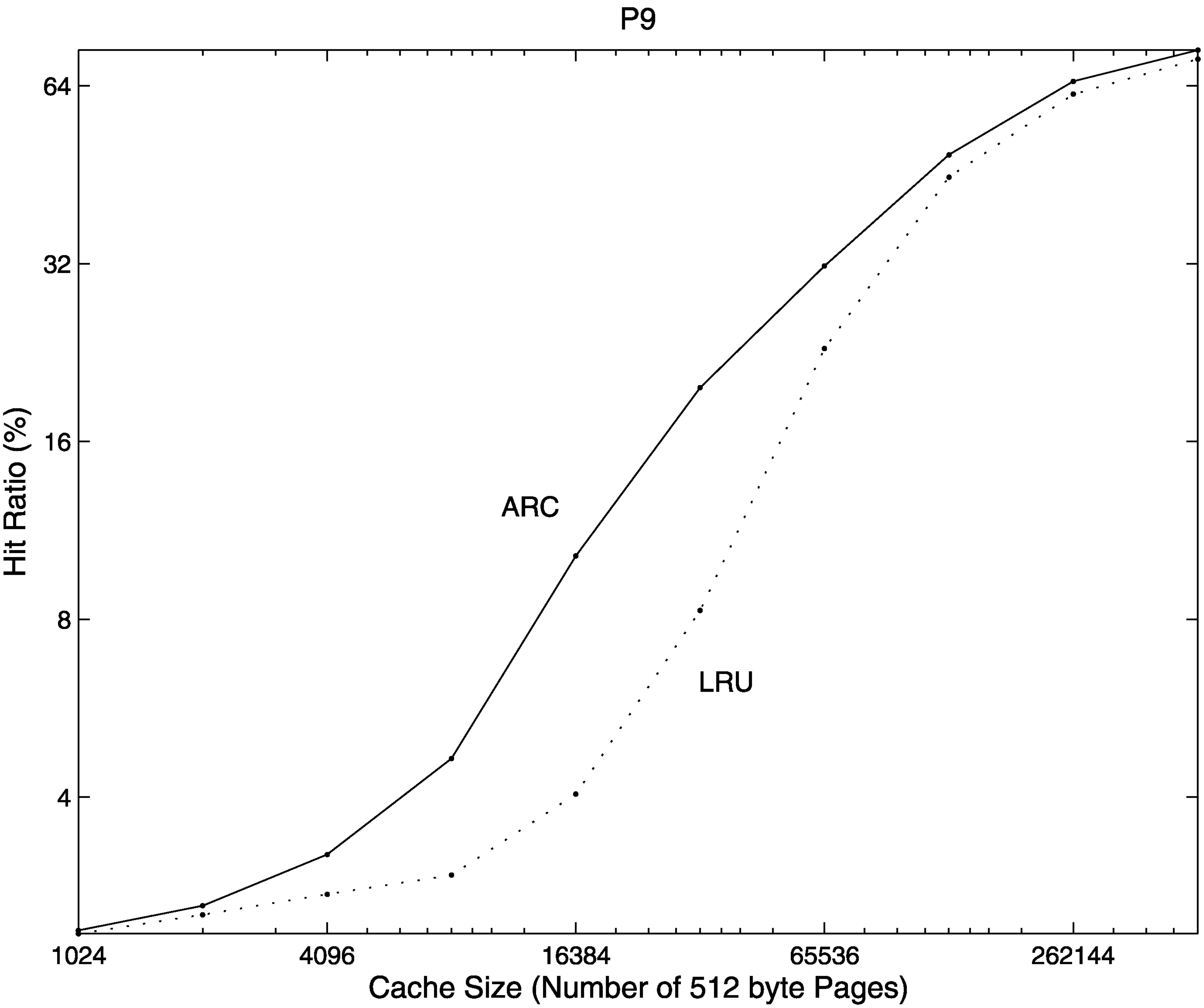

P3
Hit Ratio (%)
64
32
16
8
4
2
ARC
LRU
1024
4096
16384
65536
262144
Cache Size (Number of 512 byte Pages)

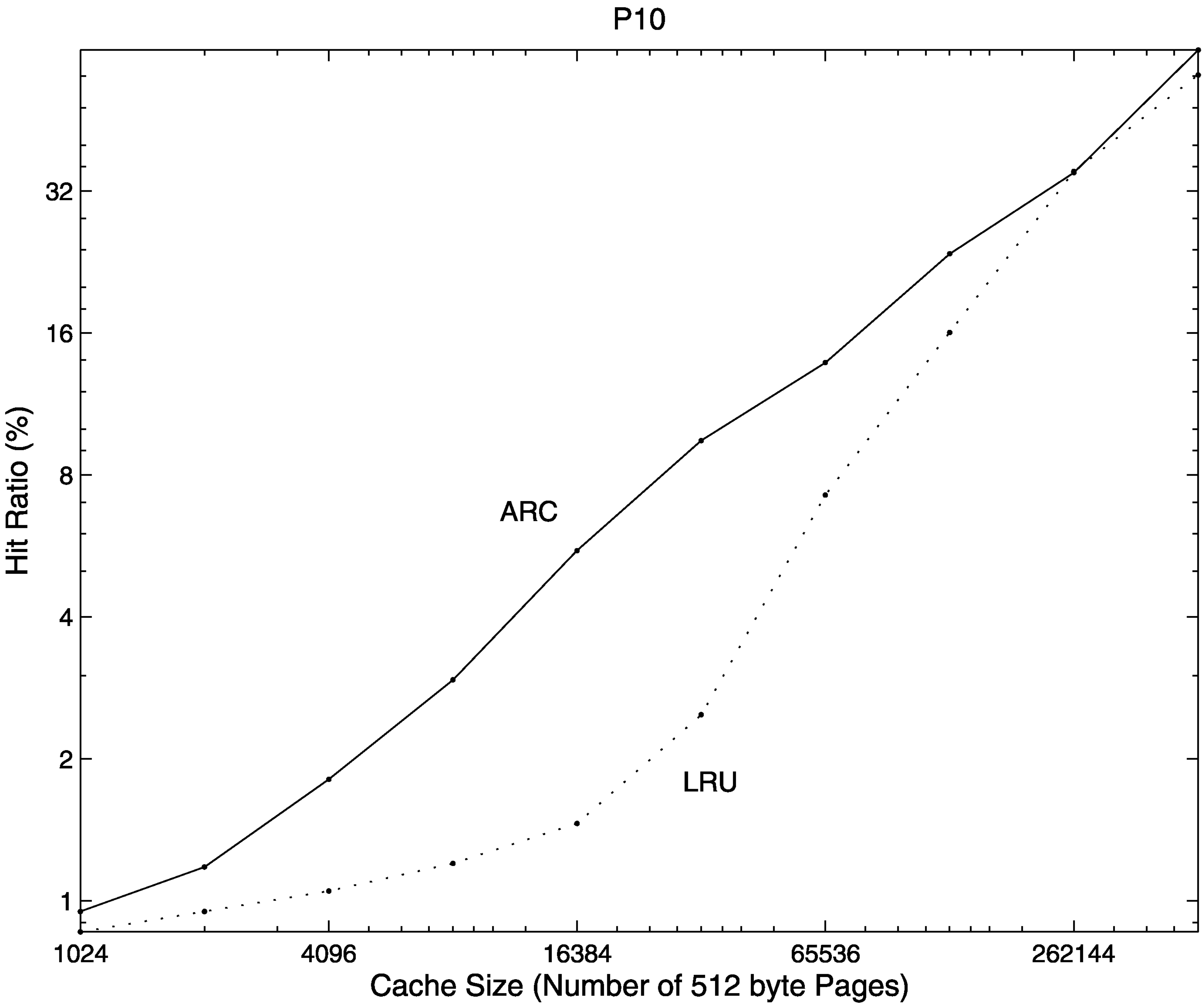

P4
Hit Ratio (%)
32
16
8
4
1024
4096
16384
65536
262144
Cache Size (Number of 512 byte Pages)
ARC
LRU

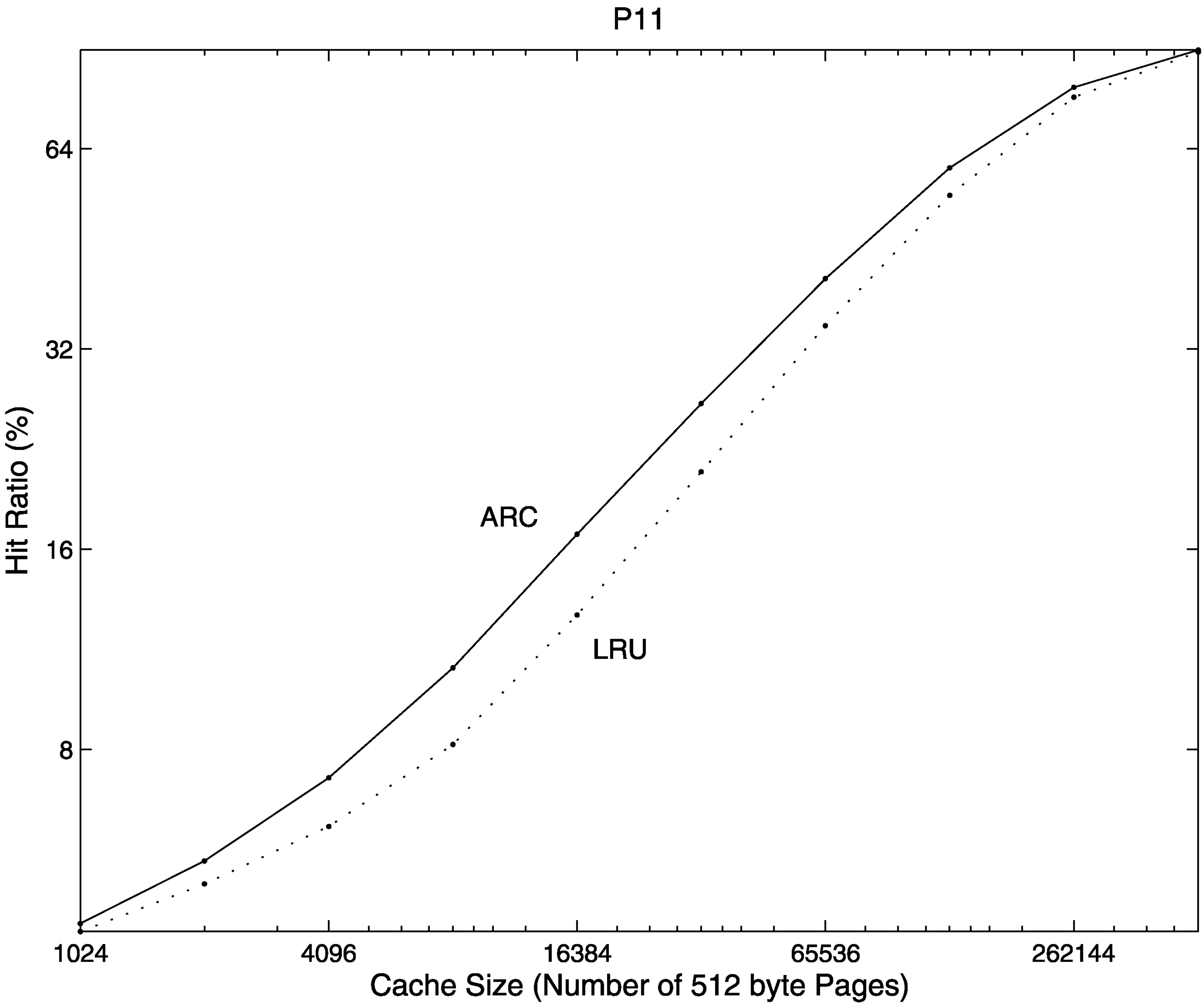

P5
Hit Ratio (%)
64
32
16
8
1024
4096
16384
65536
262144
Cache Size (Number of 512 byte Pages)
ARC
LRU

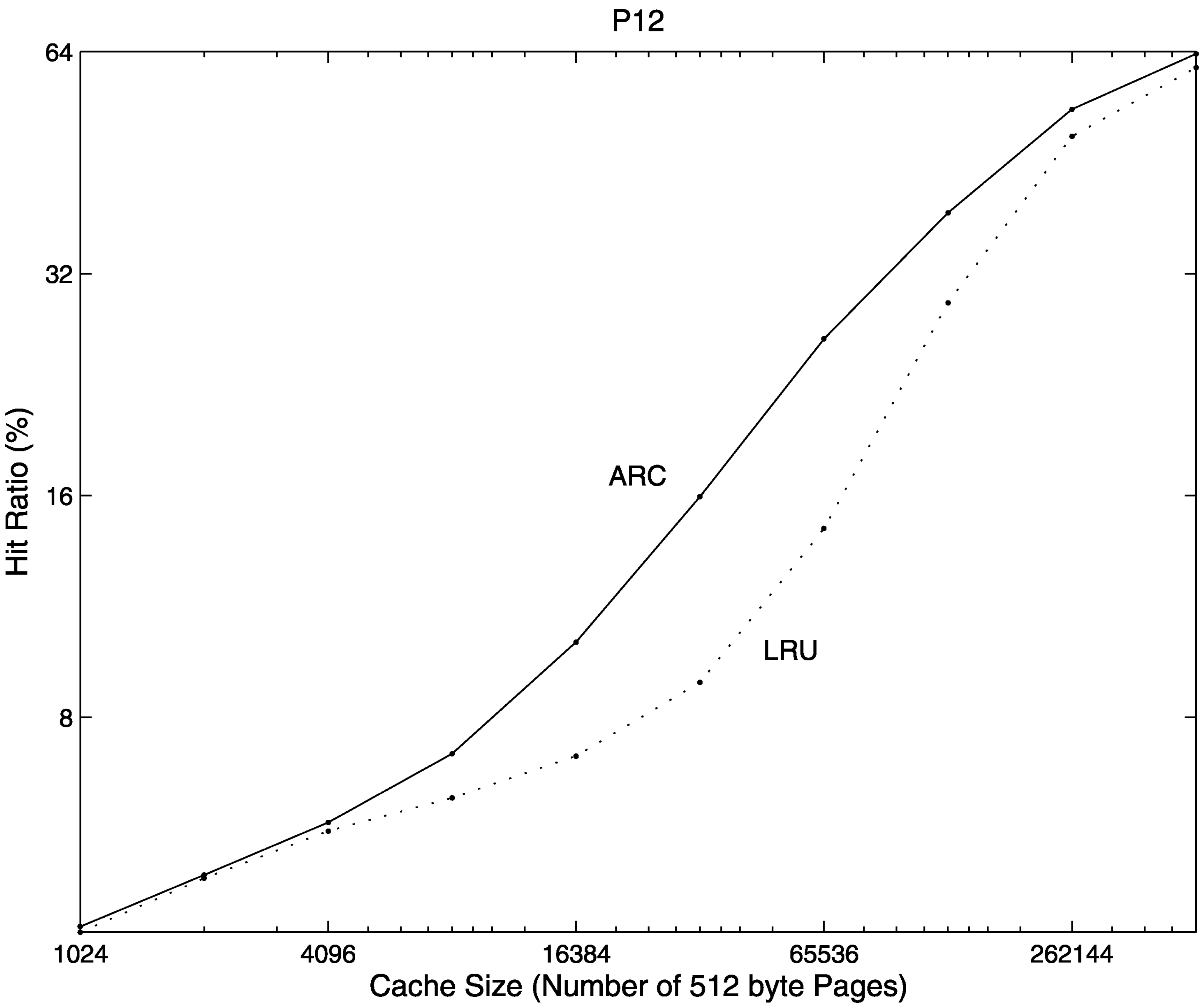

P7
Hit Ratio (%)
64
32
16
8
4
2
ARC
LRU
1024
4096
16384
65536
262144
Cache Size (Number of 512 byte Pages)

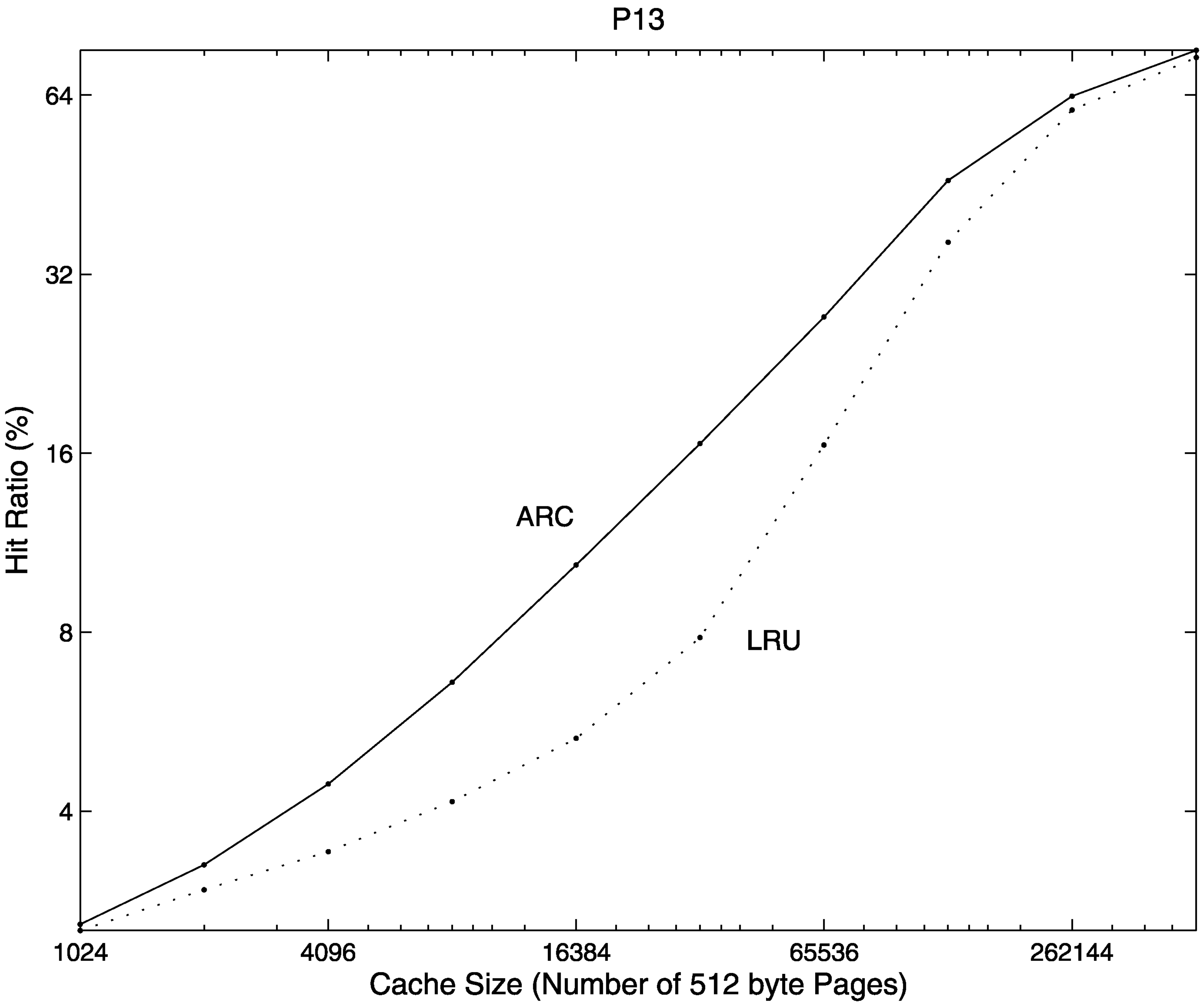

P8
Hit Ratio (%)
64
32
16
8
4
2
1
0.5
1024
4096
16384
65536
262144
Cache Size (Number of 512 byte Pages)
ARC
LRU

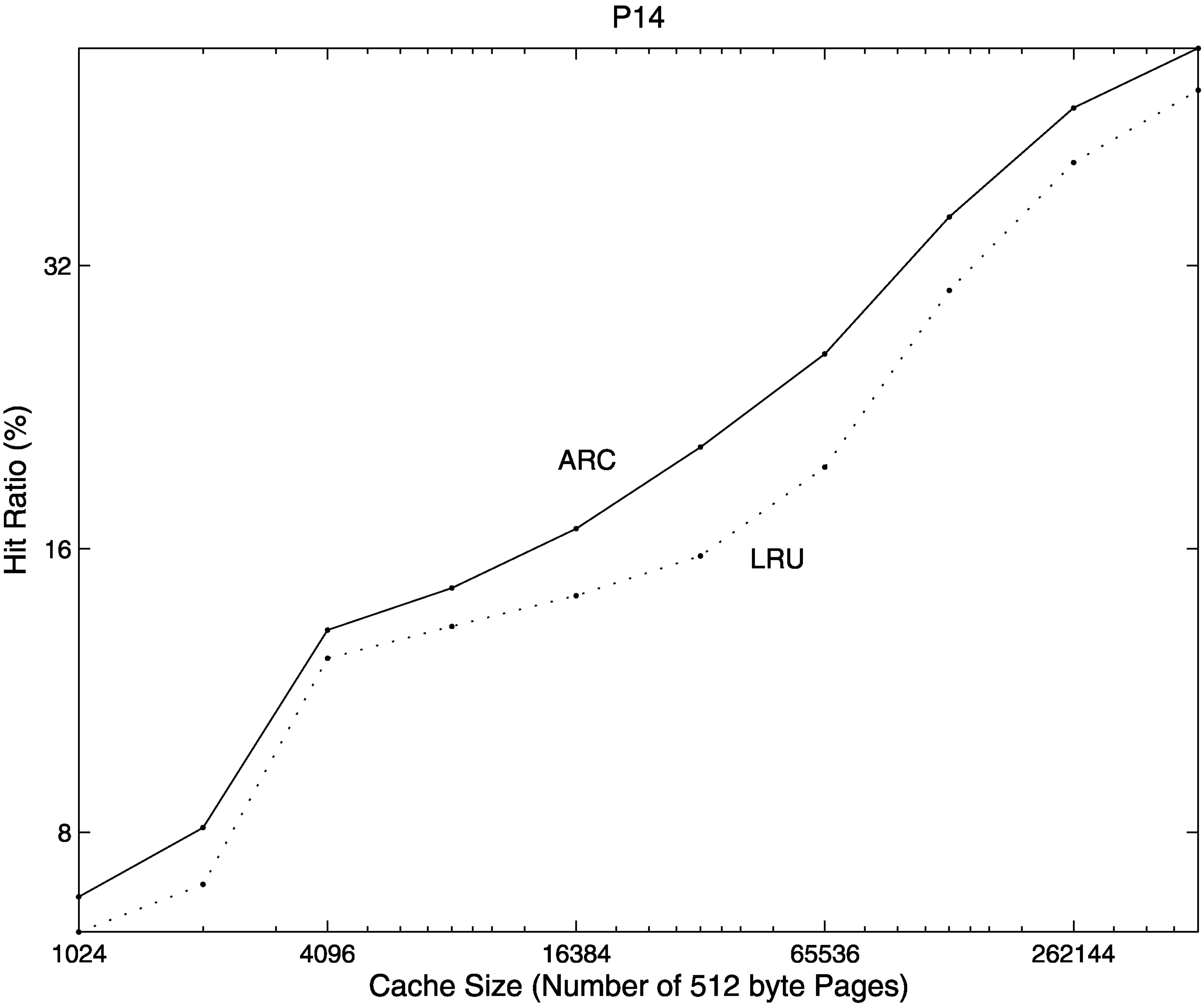

P9
Hit Ratio (%)
64
32
16
8
4
ARC
LRU
1024
4096
16384
65536
262144
Cache Size (Number of 512 byte Pages)

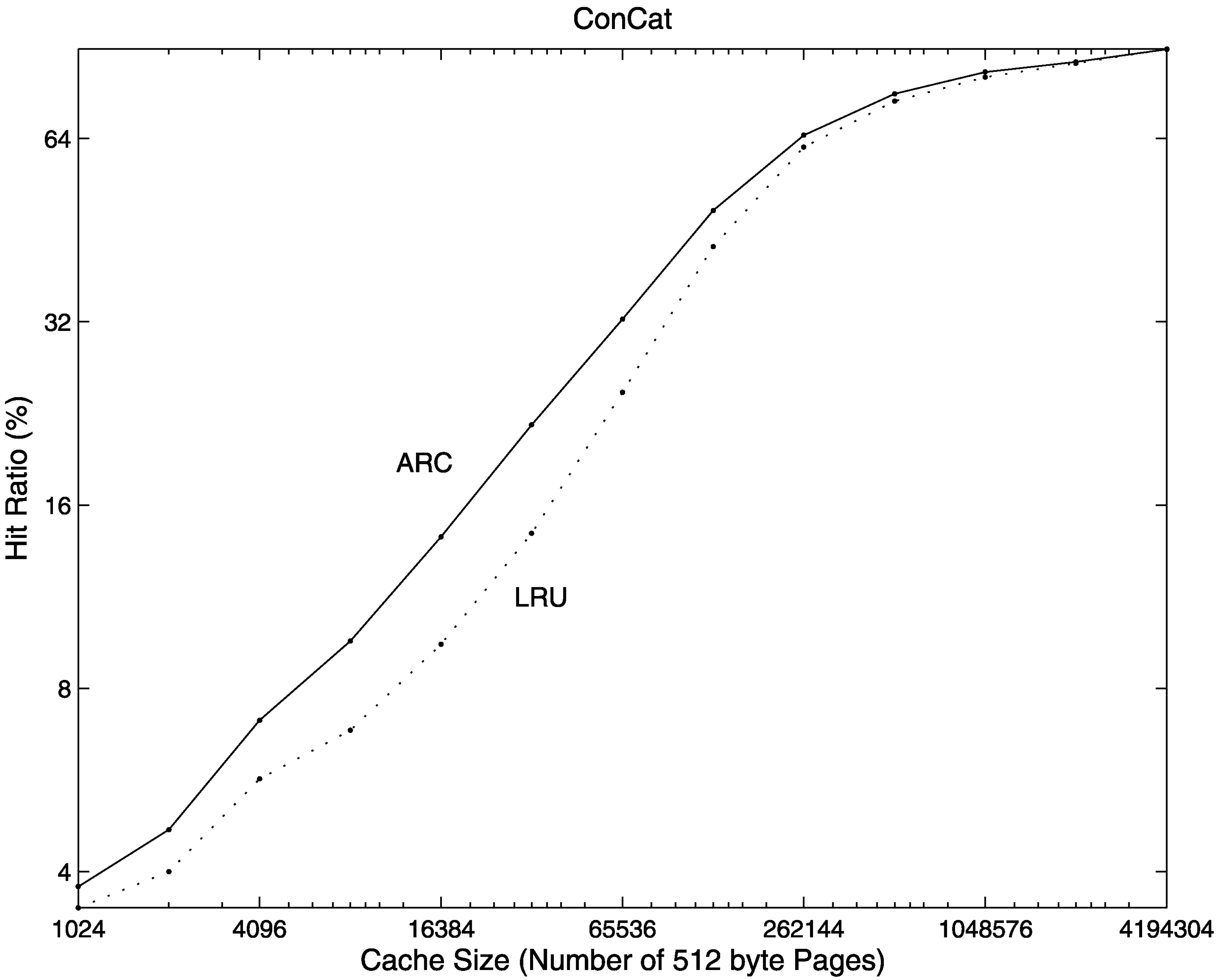

P10
Hit Ratio (%)
32
16
8
4
2
1
ARC
LRU
1024
4096
16384
65536
262144
Cache Size (Number of 512 byte Pages)

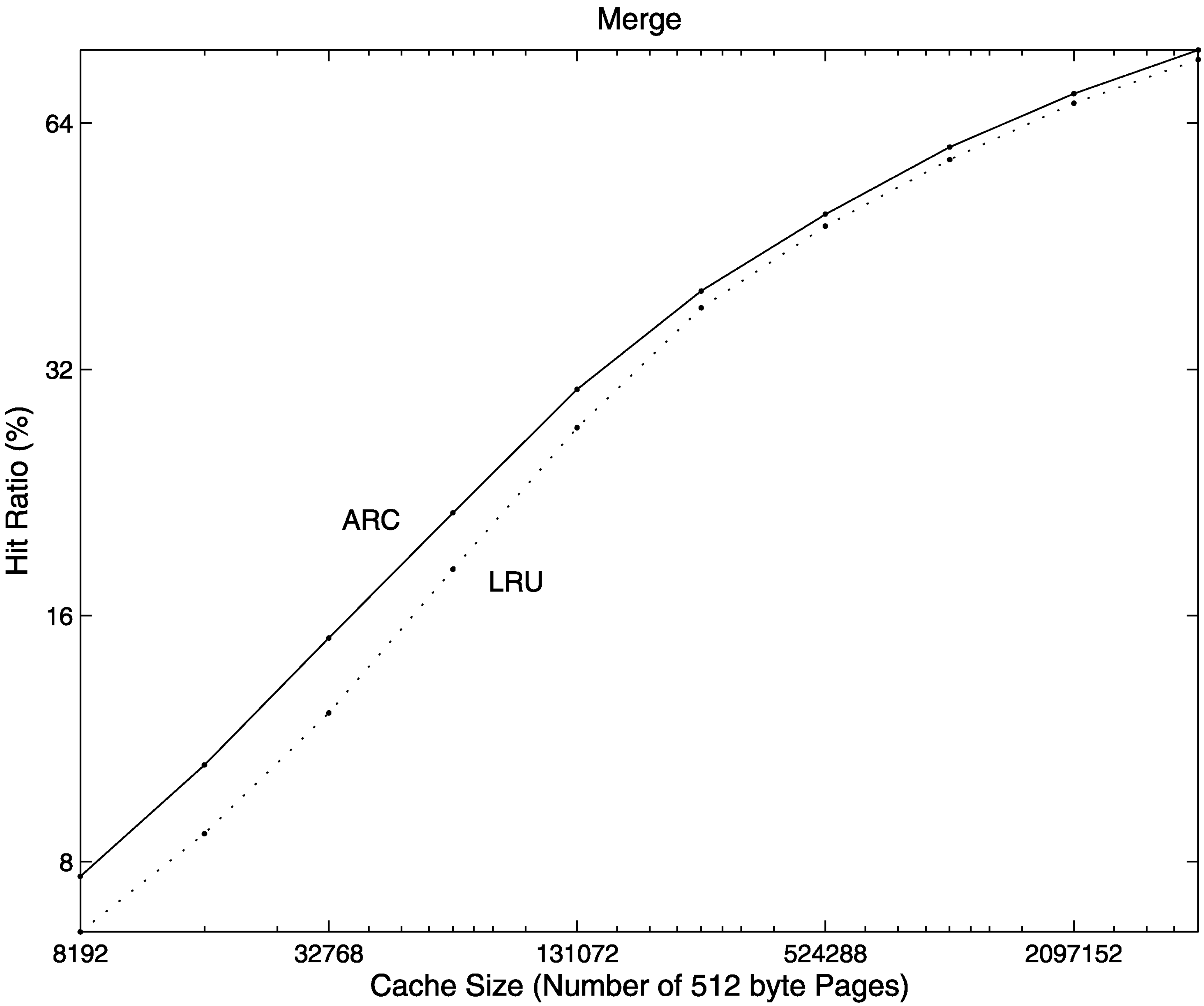

P11
Hit Ratio (%)
64
32
16
8
ARC
LRU
1024
4096
16384
65536
262144
Cache Size (Number of 512 byte Pages)


P12
64
32
16
8
Hit Ratio (%)
ARC
LRU
1024
4096
16384
65536
262144
Cache Size (Number of 512 byte Pages)


P13
Hit Ratio (%)
64
32
16
8
4
1024
4096
16384
65536
262144
Cache Size (Number of 512 byte Pages)
ARC
LRU


P14
Hit Ratio (%)
32
16
8
ARC
LRU
1024
4096
16384
65536
262144
Cache Size (Number of 512 byte Pages)


ConCat
Hit Ratio (%)
64
32
16
8
4
ARC
LRU
1024
4096
16384
65536
262144
1048576
4194304
Cache Size (Number of 512 byte Pages)


Merge
64
32
16
8
Hit Ratio (%)
ARC
LRU
8192
32768
131072
524288
2097152
Cache Size (Number of 512 byte Pages)

ARC outperforms LRU & is empirically universal

Workload	Cache Size	LRU	ARC	FRC
	MBytes	ONLINE	ONLINE	OFFLINE
P1	16	16.55	28.26	29.39
P2	16	18.47	27.38	27.61
P3	16	3.57	17.12	17.60
P4	16	5.24	11.24	9.11
P5	16	6.73	14.27	14.29
P6	16	4.24	23.84	22.62
P7	16	3.45	13.77	14.01
P8	16	17.18	27.51	28.92
P9	16	8.28	19.73	20.28
P10	16	2.48	9.46	9.63
P11	16	20.92	26.48	26.57
P12	16	8.93	15.94	15.97
P13	16	7.83	16.60	16.81
P14	16	15.73	20.52	20.55
ConCat	16	14.38	21.67	21.63
Merge	128	38.05	39.91	39.40
DS1	1024	11.65	22.52	18.72
SPC1	4096	9.19	20.00	20.11
MergeS	4096	27.62	40.44	40.18

Summary: ARC

- Low compute overhead

Summary: ARC

- Low compute overhead
- Low space overhead

Summary: ARC

- Low compute overhead
- Low space overhead
- No periodic resizing

Summary: ARC

- Low compute overhead
- Low space overhead
- No periodic resizing
- Captures both "recency" and "frequency"
  - self-tuning: autonomically adapts to evolving workloads and relentlessly balances between recency and frequency
  - empirically universal

Summary: ARC

- Low compute overhead
- Low space overhead
- No periodic resizing
- Captures both "recency" and "frequency"
  - self-tuning: autonomically adapts to evolving workloads and relentlessly balances between recency and frequency
  - empirically universal
- Scan-resistant

Summary: ARC

- Low compute overhead
- Low space overhead
- No periodic resizing
- Captures both "recency" and "frequency"
  - self-tuning: autonomically adapts to evolving workloads and relentlessly balances between recency and frequency
  - empirically universal
- Scan-resistant
- Comparable to state-of-the-art algorithms with best offline choice

Summary: ARC

- Low compute overhead
- Low space overhead
- No periodic resizing
- Captures both "recency" and "frequency"
  - self-tuning: autonomically adapts to evolving workloads and relentlessly balances between recency and frequency
  - empirically universal
- Scan-resistant
- Comparable to state-of-the-art algorithms with best offline choice
- Significantly outperforms LRU on all workloads examined

Summary: ARC

- Low compute overhead
- Low space overhead
- No periodic resizing
- Captures both "recency" and "frequency"
  - self-tuning: autonomically adapts to evolving workloads and relentlessly balances between recency and frequency
  - empirically universal
- Scan-resistant
- Comparable to state-of-the-art algorithms with best offline choice
- Significantly outperforms LRU on all workloads examined
- Requires only a handful of lines of code! See:
  - "ARC: A Self-tuning, low overhead Replacement Cache"
  - "One Up on LRU"
  - http://www.almaden.ibm.com/cs/people/dmodha

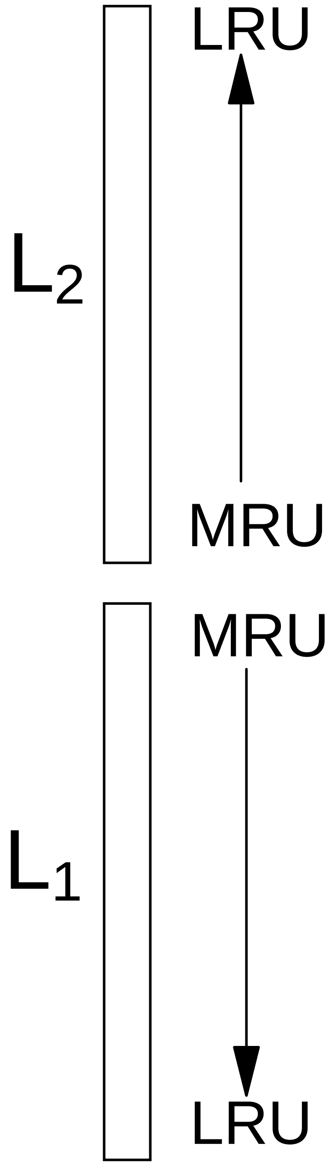
At-a-glance Comparision

	Compute Overhead	Space Overhead	Self-Tuning	Scan Resistant	No Re-sizing
LRU	constant	1x	Yes	No	Yes
LFU	log	1x	Yes	Yes	No
LRU-2	log	1x-2x	No	Depends	Yes
2Q	constant	1x-2x	No	Depends	Yes
LIRS	constant (E)	unbounded	No	Depends	Yes
LRFU	log	1x-2x	No	Depends	Yes
FBR	constant (E)	1x	No	Depends	No
ARC	constant	2x	Yes	Yes	Yes

P8: LRU, 2Q, LRU-2, LRFU, LIRS

Cache Size	LRU	ARC	2Q	LRU-2	LRFU	LIRS
	ONLINE	ONLINE	OFFLINE	OFFLINE	OFFLINE	OFFLINE
1024	0.35	1.22	0.94	1.63	0.69	0.79
2048	0.45	2.43	2.27	3.01	2.18	1.71
4096	0.73	5.28	5.13	5.50	3.53	3.60
8192	2.30	9.19	10.27	9.87	7.58	7.67
16384	7.37	16.48	18.78	17.18	14.83	15.26
32768	17.18	27.51	31.33	28.86	28.37	27.29
65536	36.10	43.42	47.61	45.77	46.72	45.36
131072	62.10	66.35	69.45	67.56	66.60	69.65
262144	89.26	89.28	88.92	89.59	90.32	89.78
524288	96.77	97.30	96.16	97.22	97.38	97.21

Cache Directory (Registry)



```
if (L1->hit(page)) {
        L1->delete(page);
        L2->insert_mru(page);
  }
else if (L2->hit(page)) {
        L2->delete(page);
        L2->insert_mru(page);
  }
else if (L1->length() == c) {
        L1->delete_lru();
        L1->insert_mru(page);
  }
else {
        if (L1->length() + L2->length() == 2*c) {
               L2->delete_lru();
        }
        L1->insert_mru(page);
  }
```

Applications

- Storage controllers
- File Systems
- Operating Systems
- Disks, RAID
- Databases
- Middleware
- Web Caching, Search Query Caching
- Micro-Processors
- Data Compression

ARC outperforms LRU

Workload	Cache Size	LRU	ARC
	MBytes	ONLINE	ONLINE
P1	16	16.55	28.26
P2	16	18.47	27.38
P3	16	3.57	17.12
P4	16	5.24	11.24
P5	16	6.73	14.27
P6	16	4.24	23.84
P7	16	3.45	13.77
P8	16	17.18	27.51
P9	16	8.28	19.73
P10	16	2.48	9.46
P11	16	20.92	26.48
P12	16	8.93	15.94
P13	16	7.83	16.60
P14	16	15.73	20.52
ConCat	16	14.38	21.67
Merge	128	38.05	39.91
DS1	1024	11.65	22.52
SPC1	4096	9.19	20.00
MergeS	4096	27.62	40.44