

TM

TM on Haswell

- RTM
 - xbegin / xend 命令でトランザクション部をはさむ
- HLE
 - xacquire / xrelease プレフィックスを既存のロック取得 cmpxchg やmovにつけるだけ
 - Lock Elision専用

RTM Lock Elision Example

.retry:

xbegin

.transaction_failed

mov eax, dword [lock]

test eax, 0x1

jz .transaction

xabort 0xff

.transaction:

; transaction

xend

.transaction_failed:

条件を見てjmp retryするかlock取得

HLE Example

.get_lock:

mov eax, 0x1

xacquire xchg eax, dword [lock]

test eax, eax

jnz **.get_lock**

; transaction

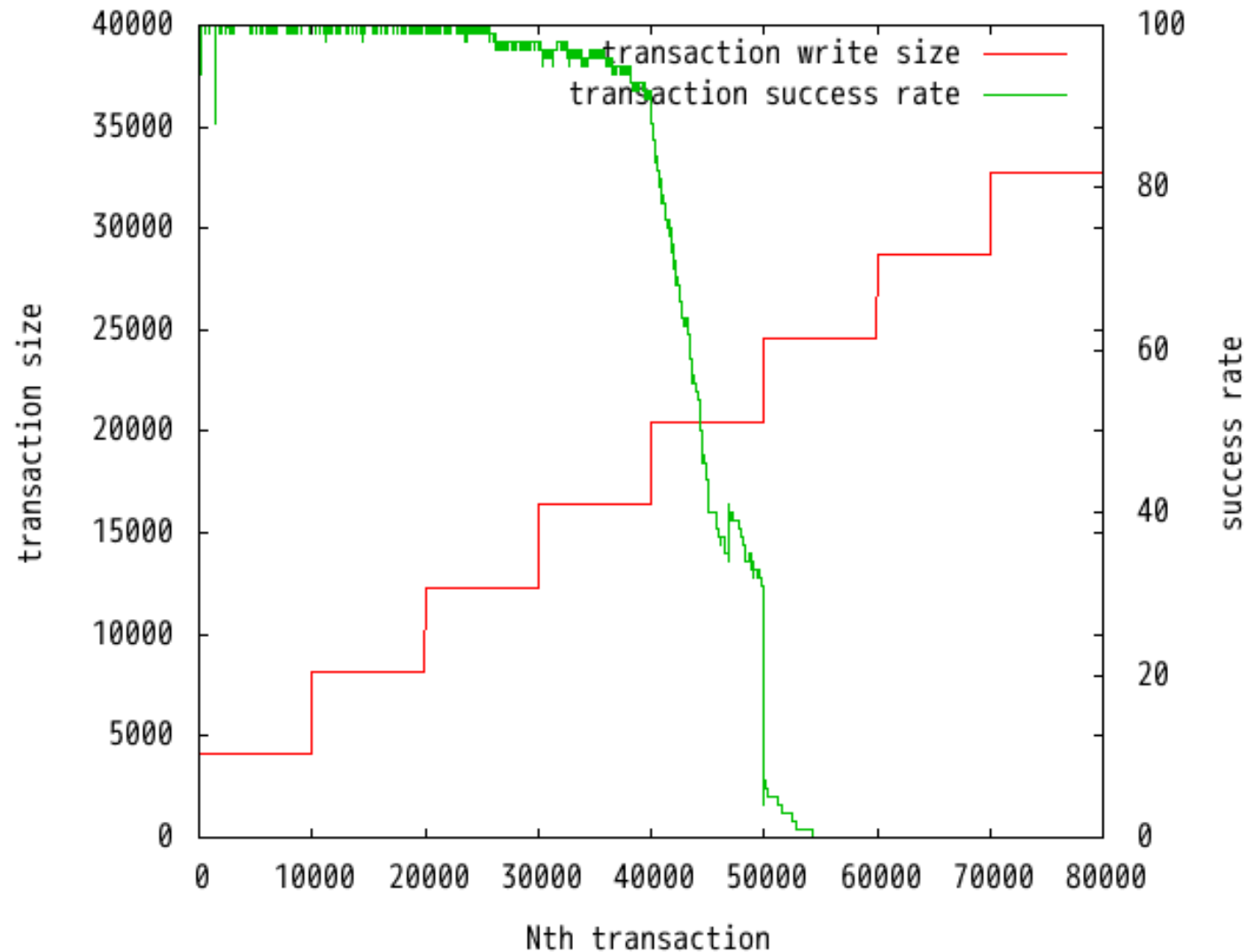
xrelease mov 0, dword [lock]

Haswell TM parameters

- Read-set size 4M (i7) or 6.5M (Xeon) = 実測
 - LLCまで追っているが、これより大きいと、トランザクション成功予測機構が失敗を予測
 - トランザクション突入直後に殺される
- Write-set size
 - 24K位まで (実測)
 - L1=32KBだが、Readと同様の状況

トランザクション成功予測機構

- 存在はマニュアルには書いていない



トランザクションのオーバーヘッド

- 空のトランザクション ~30 ns
- トランザクショナル・アボート
 - 1000 ns (アボート予測成功時)
 - 8000+ ns (アボート予測失敗時)

STAMP

- Lock Elisionを使っている
- STAMPをRTM/HLEに移植した
- RTM実装では失敗時のリトライ回数やリトライするかどうかをチューニングした
- HLEはそういうインターフェースは無い
 - でも使うのは簡単

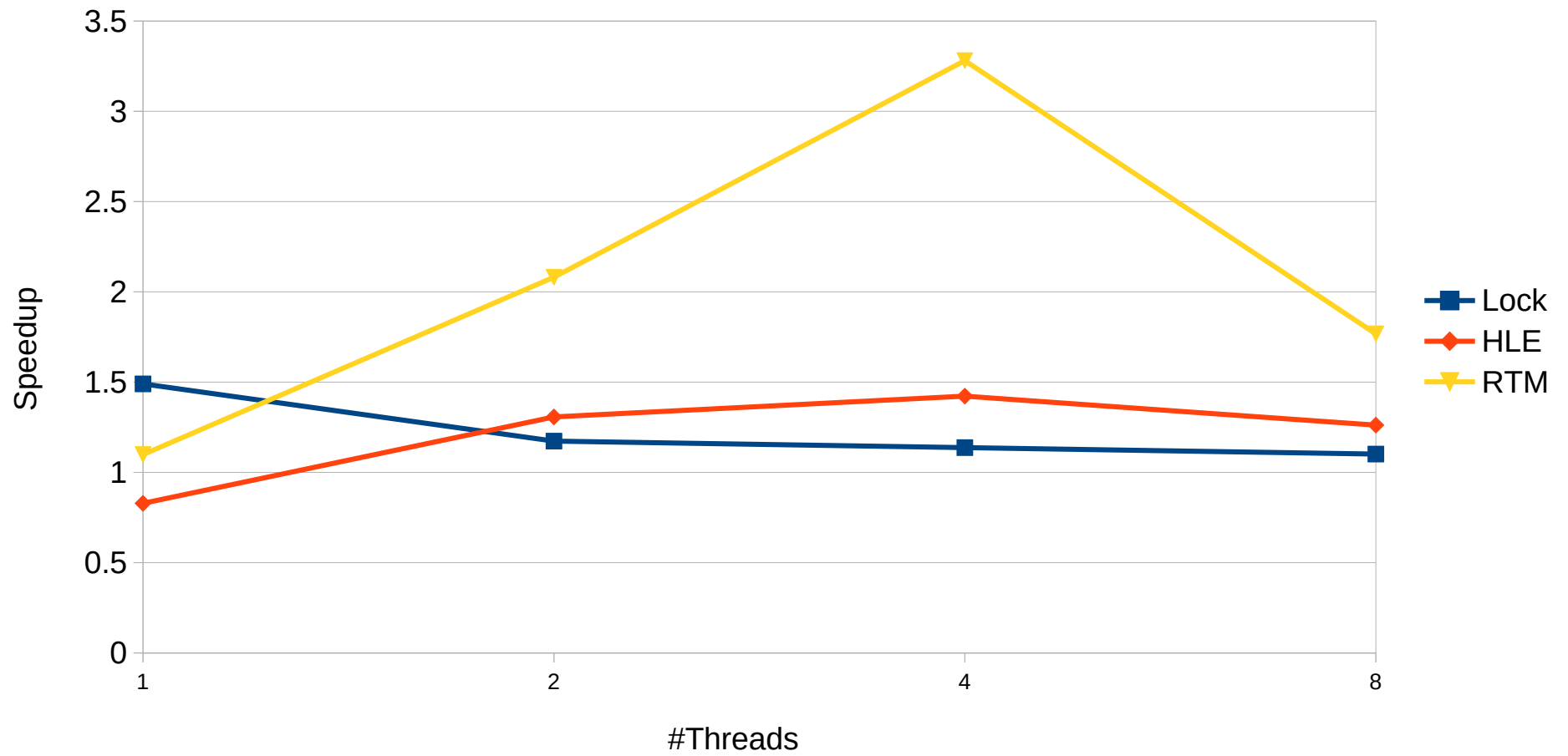
RTMのチューニングパラメータ

- 理由を問わず(Intelのみ) リトライは16回まで
 - RTMではPersistent AbortでなくともRetryビットが立っていなかったり
 - Capacity理由でないのにCapacityビットが立っていたりするため
- その他色々

測定環境

- Core i7 4770S 3.10 GHz
 - Turbo boost機能はオフ
 - 動的なコアの逡倍率変化もオフ

GENOME



GENOME アボート理由

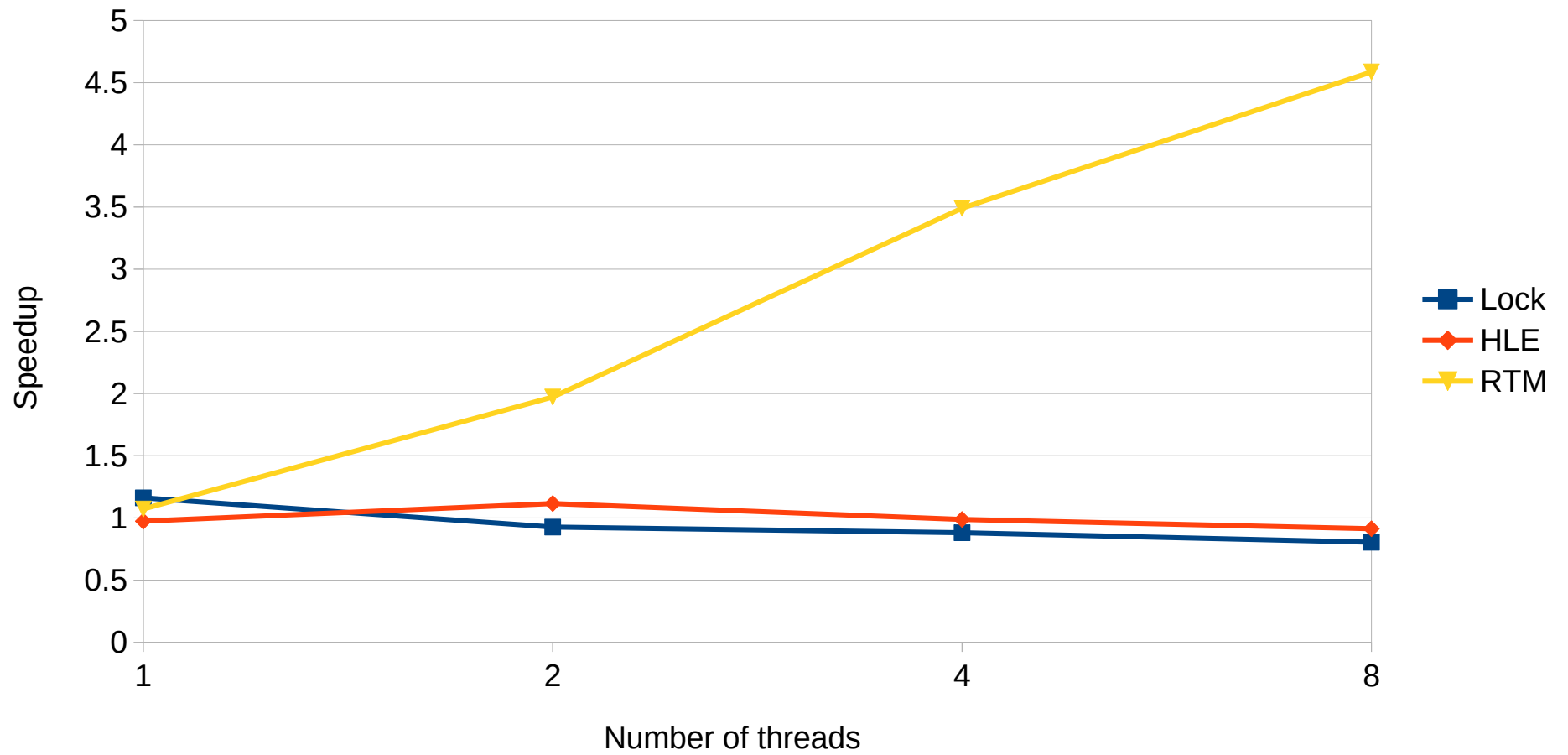
- 4 threads

2489220	tx_enter
2467969	tx
0	global_lock_acquired
641006	abort
0	explicit
116588	retry
116842	conflict
351768	capacity
0	debug
0	nested

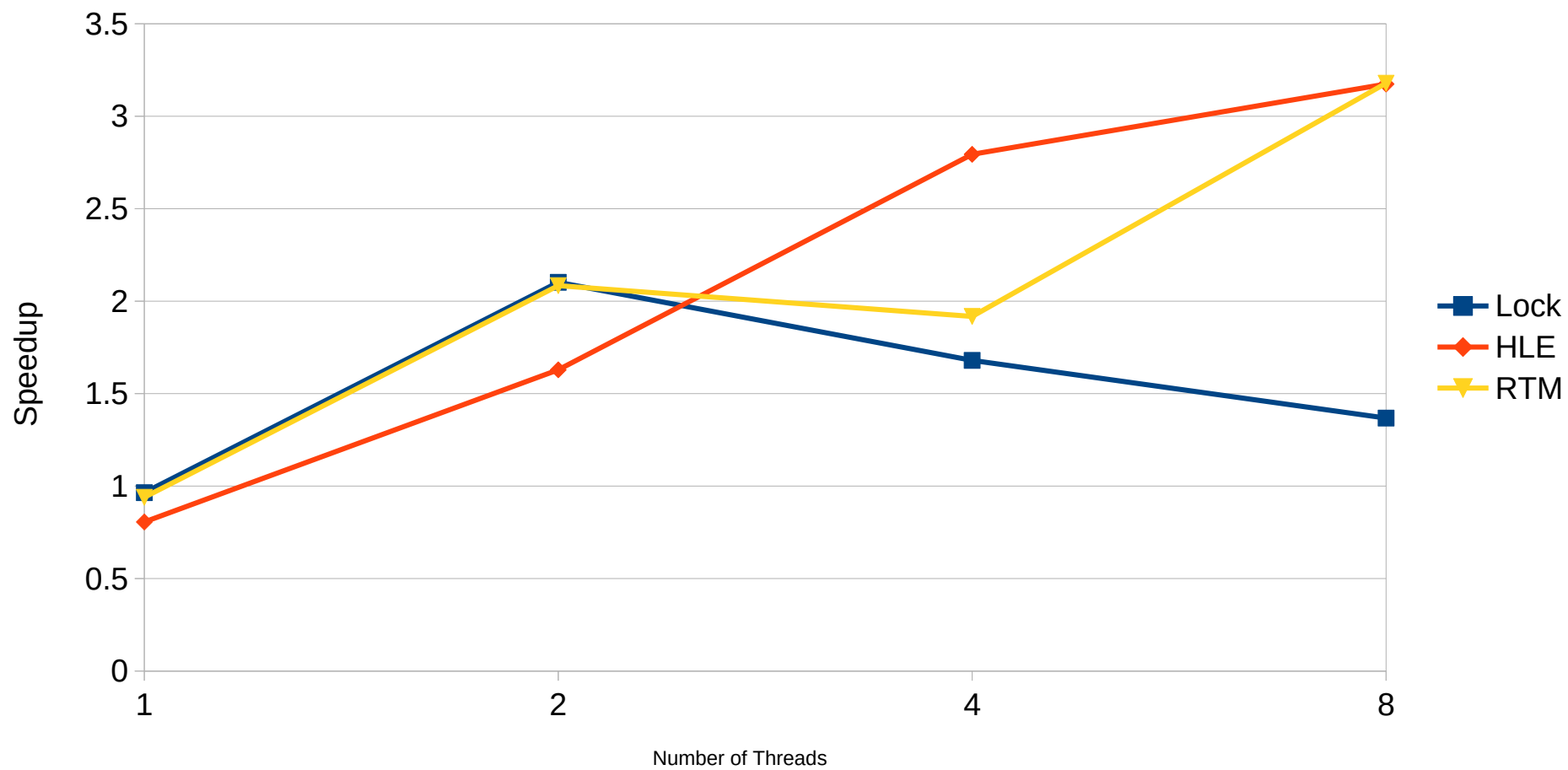
- 8 threads

2489220	tx_enter
2192610	tx
0	global_lock_acquired
9763869	abort
0	explicit
3874396	retry
3875316	conflict
5710884	capacity
0	debug
0	nested

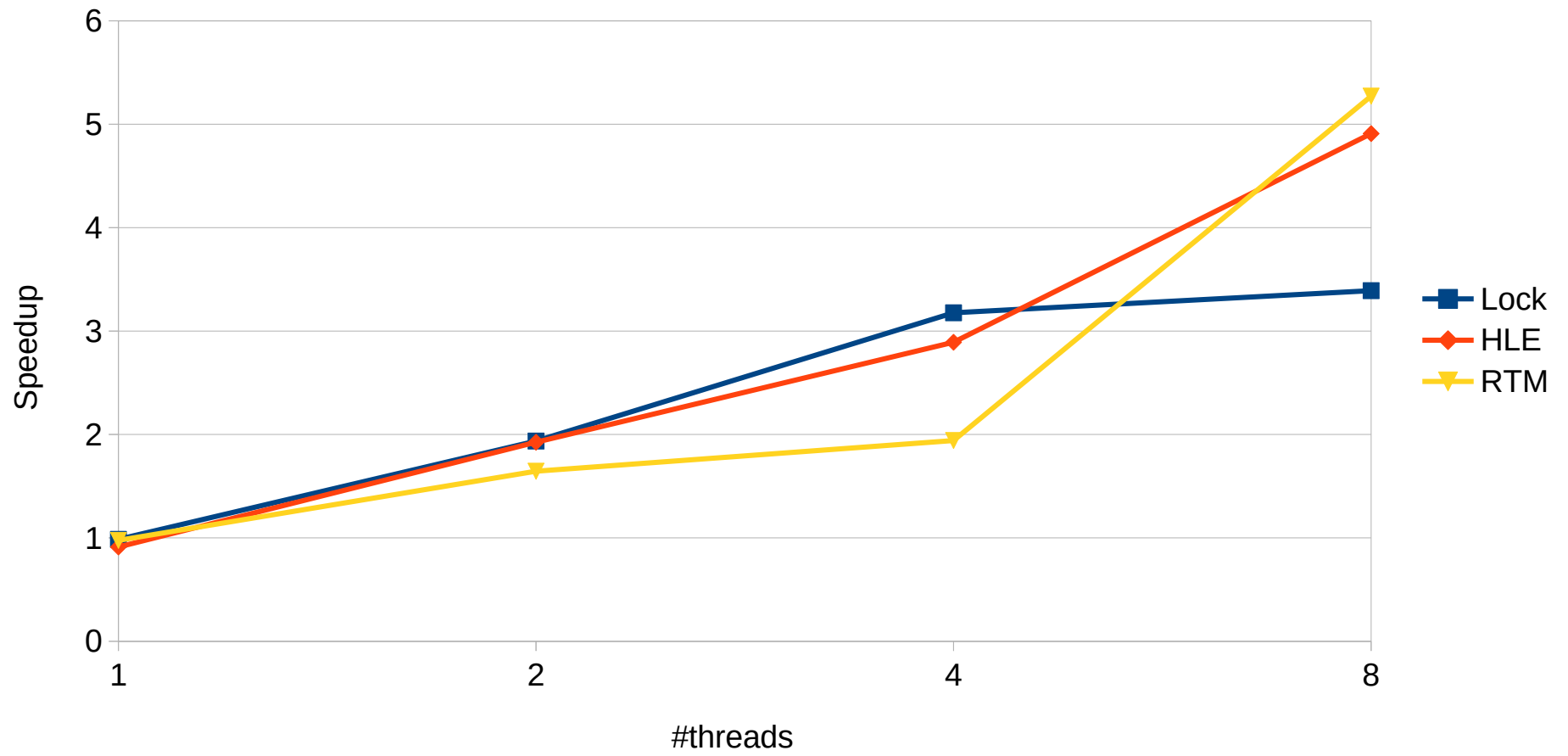
Intruder



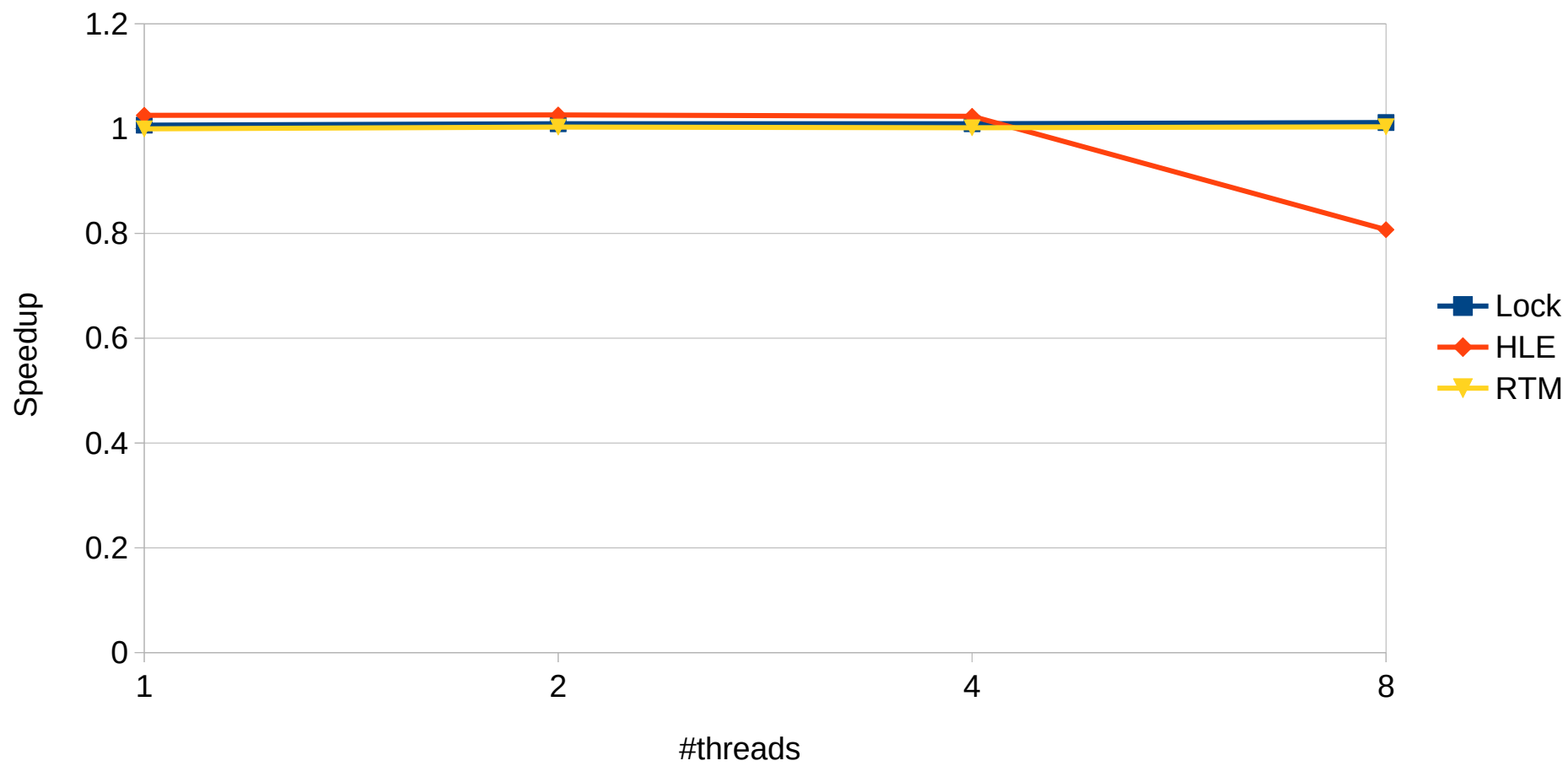
KMEANS-HIGH



KMEANS-LOW



Labyrinth

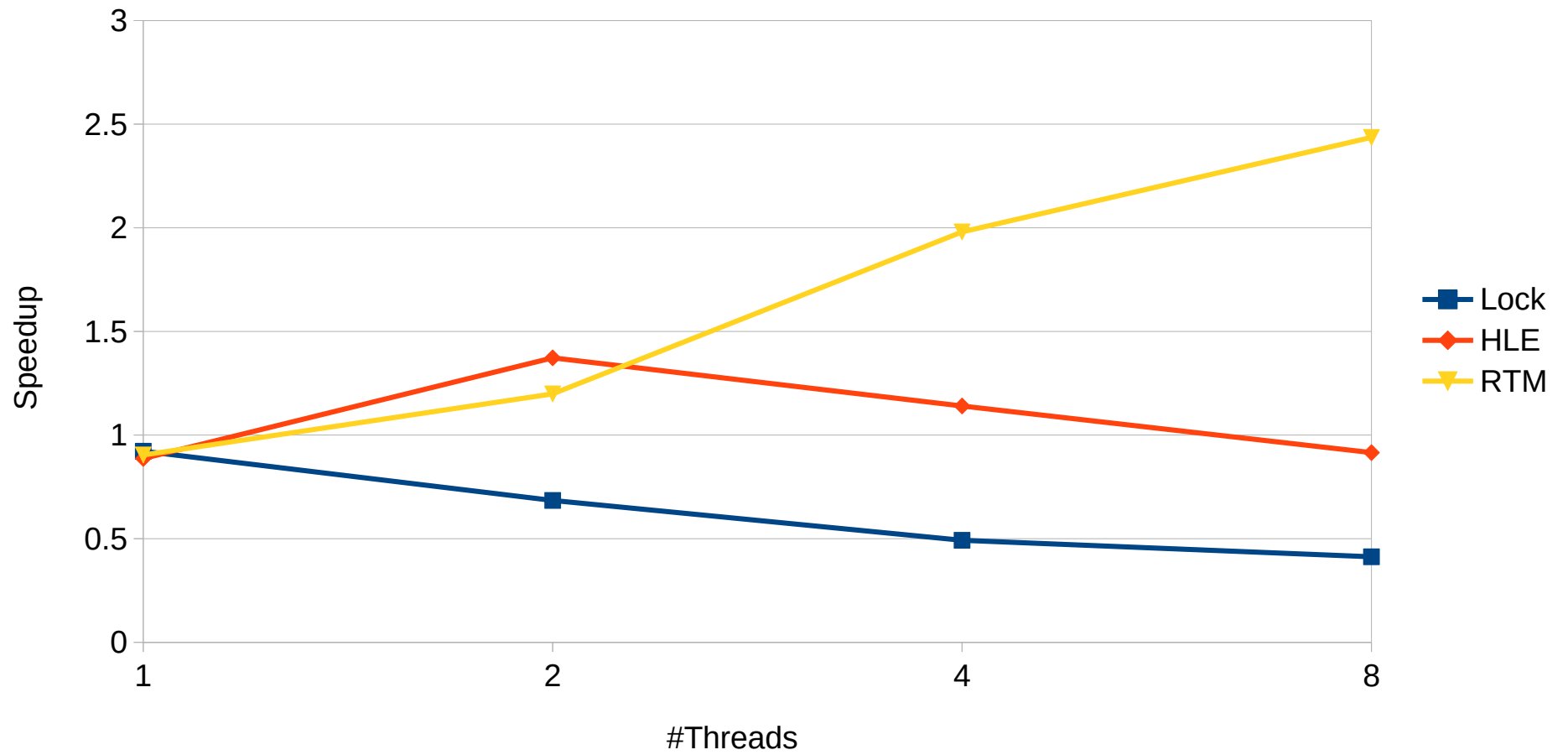


Labyrinth (8 threads)

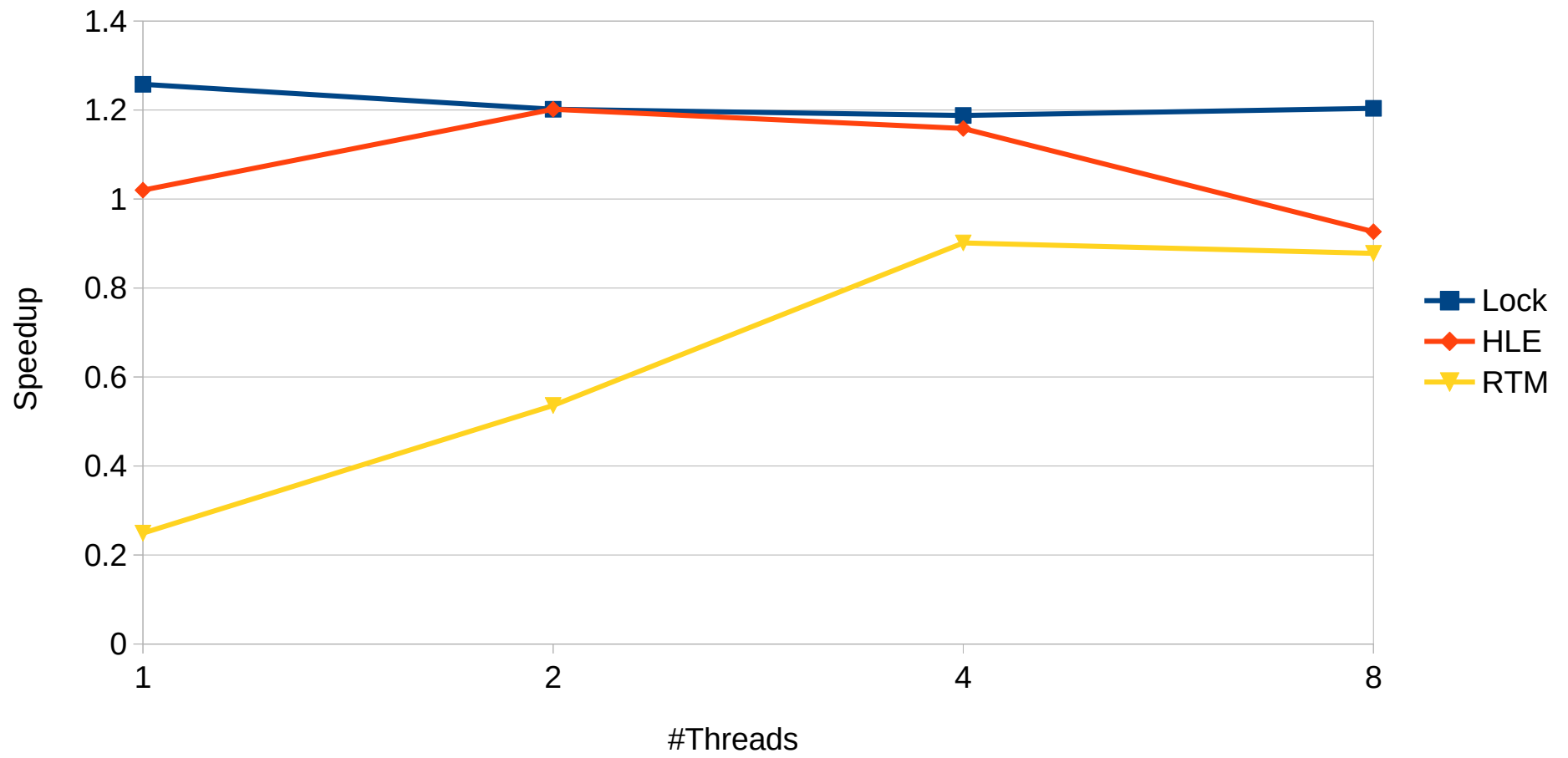
- 9回目で成功する
- 半数はロックになる

1040	tx_enter
521	tx
0	global_lock_acquired
8824	abort
0	explicit
28	retry
28	conflict
19	capacity
0	debug
0	nested

SSCA2



Yada



Yada (4 threads)

- 平均4回リトライして5回目で成功
- アドレスコンフリクトしてる

2574594	tx_enter
2161431	tx
0	global_lock_acquired
8040193	abort
0	explicit
5564264	retry
5564342	conflict
2431261	capacity
0	debug
0	nested

アボート率を減らせば性能向上？

- アボートするのは大体決まったリージョンのトランザクション
 - 最初からロックにすることでアボートのオーバーヘッド減らせる？
 - Delinquent transaction除去
- アボート率は減ったが、性能向上せず
 - ロックになっているのだから当たり前

結論

- HLEはチューニング効かない
 - RTMを使うべき
- 予測機能の機嫌次第ではアボートされる
 - 非常に厄介