Multiprocesorski sistemi

"*Snoopy*" protokoli

Milo Tomašević

SI4MPS

"*Snoopy*" protokoli

- Relativno lako rešenje problema koherencije
  - Nadgradnja kontrolera keš memorije na prirodan način
    - Nadgledanje, promene stanja
  - Korišćenje osobina magistrale
    - "*Broadcast*" transakcije, serijalizacija
- WTI
  - 2 stanja: Invalid, Valid
  - Nema novih transakcija i žica na magistrali
  - Vrlo loše performanse !
- Za bolje performanse => odloženi upis (WB)

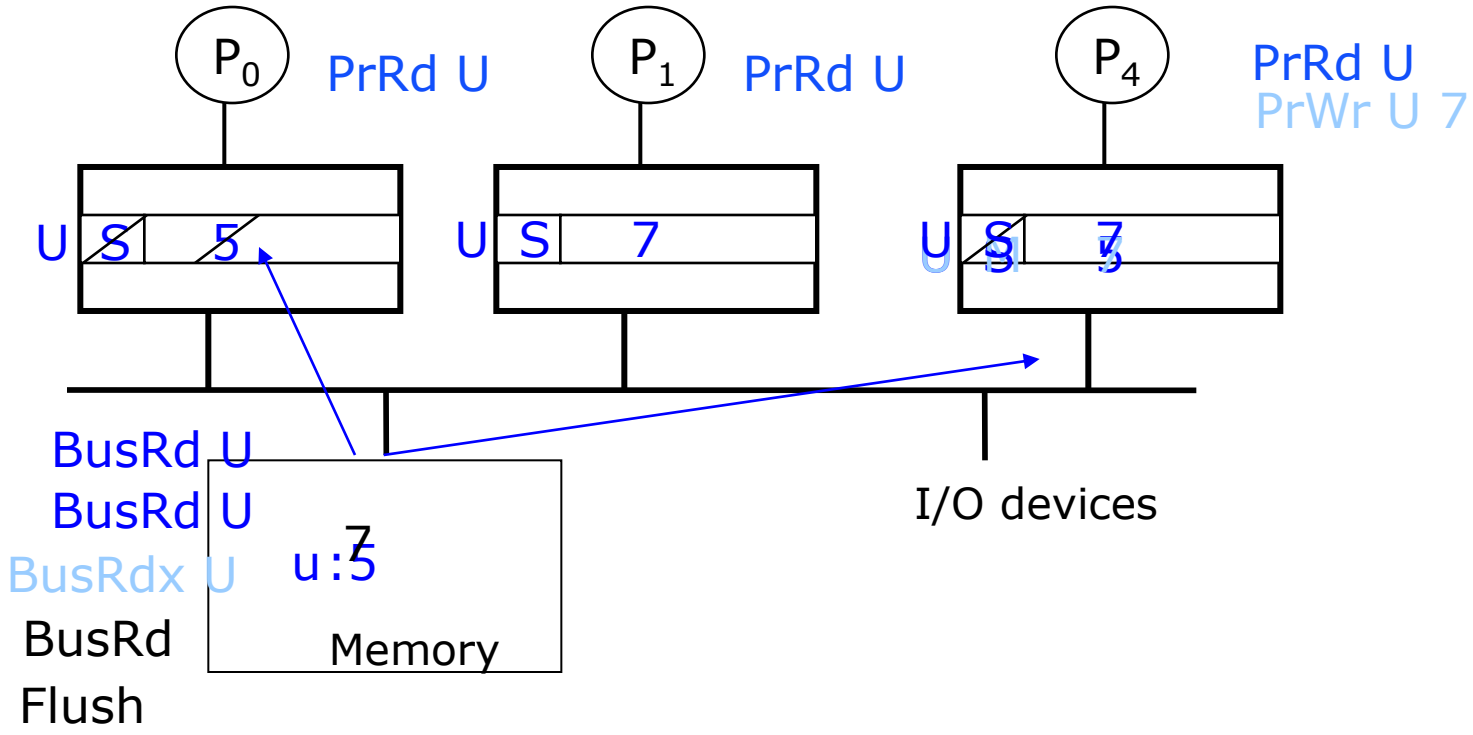
Trenutni ili odloženi upis

- WT
  - Svaki upis vidljiv na magistrali
  - Produžava latenciju
  - Potreban veliki propusni opseg, osobito u multiprocesorima
  - U proseku 15% pristupa keš memoriji - upisi
- WB
  - Samo prvi upis ide na magistralu i poništava ostale kopije
  - Ostali upisi u nizu od istog procesora lokalni
  - Protokol zahteva treće stanje - vlasništvo (jedina kopija!)
  - Samo vlasnik može da upisuje
  - Ako hoće da upiše, a nije vlasnik,
    mora poništiti druge kopije

MSI protokol – stanja i transakcije

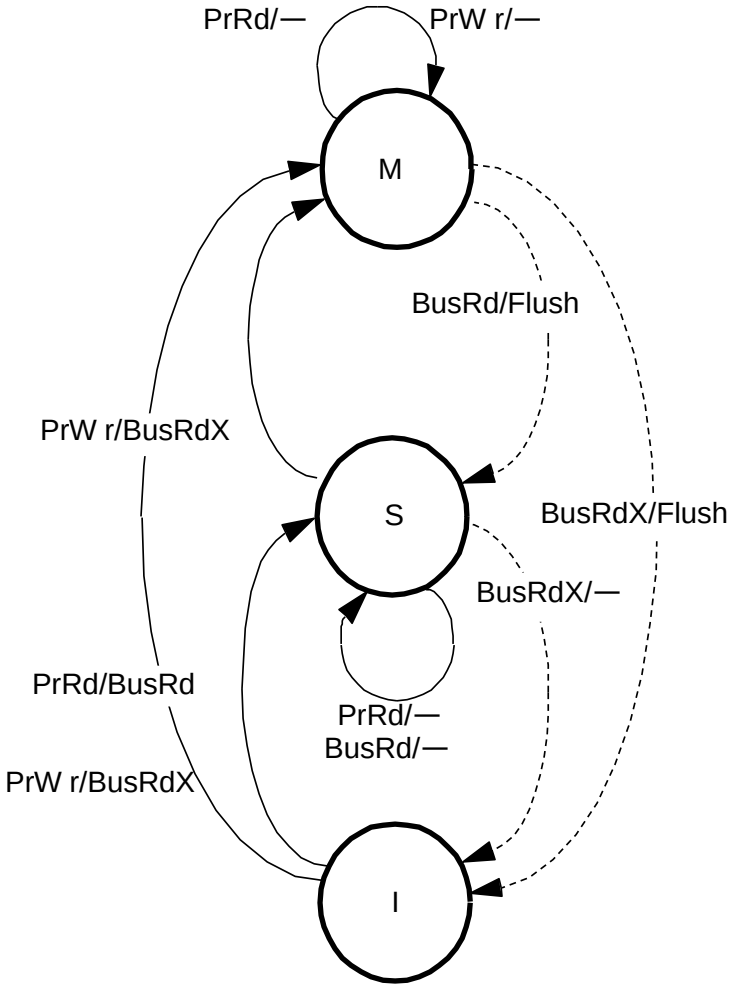
- Stanja
  - Invalid (I) – nevažeći ili blok nije prisutan
  - Shared (S) – važeća kopija, memorija ažurna, može, ali ne mora biti drugih kopija
  - Modified (M) – jedina važeća kopija, memorija neažurna, vlasništvo, ekskluzivnost
- Operacije procesora
  - PrRd – čitanje
  - PrWr - upis
- Transakcije na magistrali (prenose čitav blok)
  - Čitanje (BusRd) – iz memorije ili drugog keša
  - Ažuriranje memorije - Write-back (Flush)
  - Čitanje sa namerom za upis (BusRdX)
    - Poništavanje ostalih kopija

MSI Protokol - primer



MSI protokol - dijagram stanja

- RH – bez promene, lokalno
- RM dobija blok u stanju S
  - ... čak i ako je jedina kopija !
  - M -> C ili C -> C, M
- WM ostavlja blok u M
  - BusRdX
  - Postaje vlasnik
- WH u S se tretira kao WM
  - Ignoriše poslate podatke
  - Može i BusUpgr (bez podataka)
  - Upis u M – lokalno!
- Zamena
  - Ako je u M, mora "*write-back*"



MSI protokol: dokaz koherencije

- Upisi i čitanja se mogu obavljati bez izlaska na magistralu - koherencija garantovana?
- Propagacija upisa (“vidljivost”)
  - Upis u nemodifikovan blok vidljiv kroz BusRdX
  - Podaci se dobiju naknadnim invalidacionim promašajem
  - Tako postaju vidljivi i upisi u modifikovan blok
- Serijalizacija upisa (svi ih vide u istom poretku)
  - Upisi koji generišu BusRdX - uređeni magistralom na isti način u odnosu na sve procesore
    - obave se u keš memoriji pre drugih transakcija
  - Lokalni upisi – između dve transakcije mogući samo u ekskluzivnu kopiju (M)
    - ... pa su zato u programskom poretku (kao i lokalna čitanja)
    - R i W od ostalih procesora ih vide nakon transakcije

MSI protokol

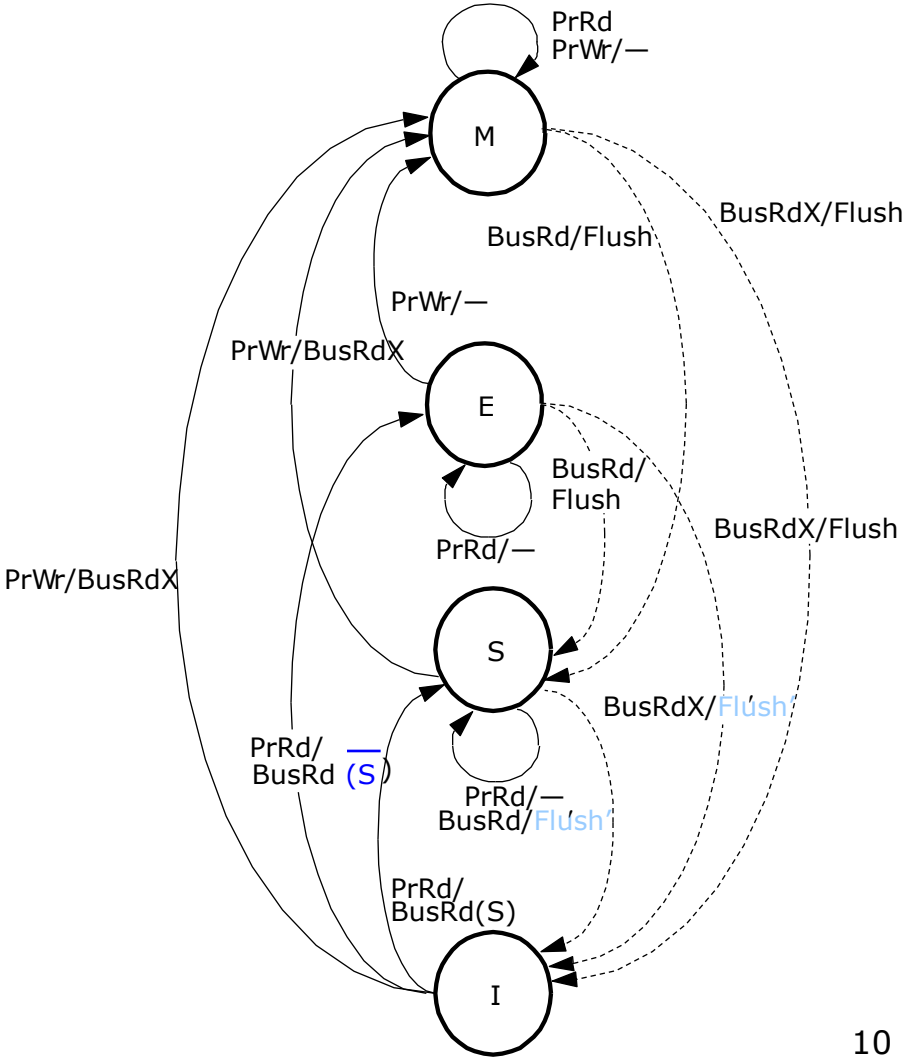
- Šta uraditi na BusRd u stanju M ?
  - Ako se očekuje dalje čitanje, M -> S (*read sharing*)
  - Ako se očekuje migratorna deljivost, M -> I
- Problem – invalidacioni promašaji
  - Rešenje – validacija bloka
  - *Read Broadcast* (*Read Snarfing*)
  - Samo jedan invalidacioni promašaj po bloku
  - Povećana interferencija keša
- Problem – čitanje, pa upis u blok koji nije deljen
  - Dve transakcije: BusRd (I -> S), pa BusRdX (S->M)
  - Neefikasno čak i kad nema deljenja
  - Nema problema kod jednoprocesorskih
  - Razdvaja se ekskluzivnost od vlasništa, novo stanje!

MESI protokol

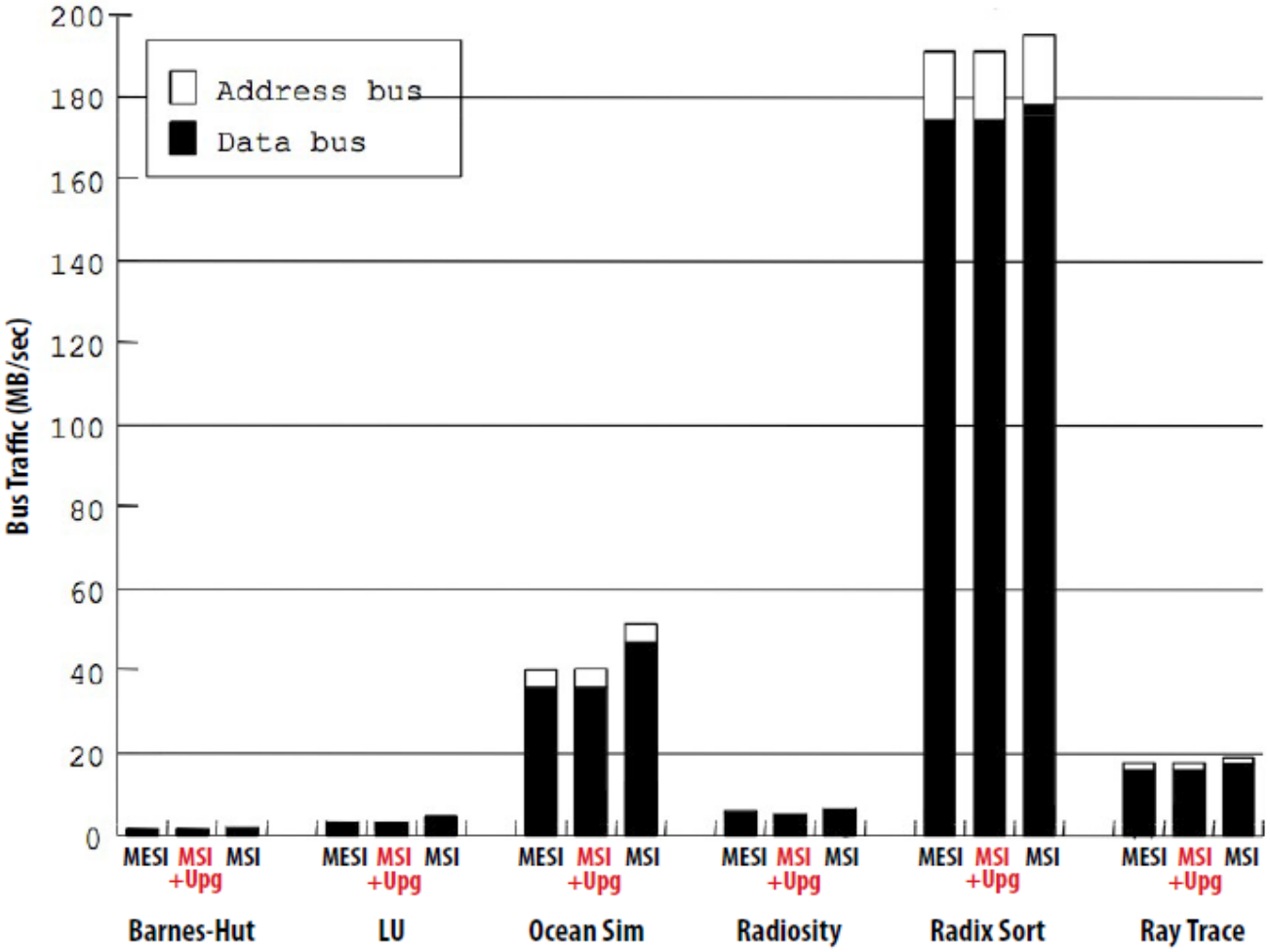
- Nadgradnja MSI protokola
  - Exclusive (E) – jedina važeća kopija, memorija ažurna
  - Upis u ovom stanju ne zahteva izlaz na magistralu, već samo ažuriranje stanja
  - Stanje S sada, praktično, označava pravo deljenje
  - Illinois (Papamarcos, Patel)
  - Korišćen u mnogim komercijalnim procesorima
- Dinamička detekcija deljivosti pri čitanju
  - Novi signal na magistrali (S)
  - Realizacija - "ožičeno ILI" (*wired-OR*)
  - Svi koju imaju blok podižu ovaj signal
  - Blok se prihvata u S ili E u zavisnosti od stanja signala

MESI Protokol - dijagram stanja

- Pri promašaju
  - Direktan prenos C -> C
  - Ažuriranje memorije
- WH u stanju E efikasniji
- Zamena
  - Nevidljiva na magistrali
  - Može ostaviti u stanju S samo jednu kopiju



MSI vs. MESI



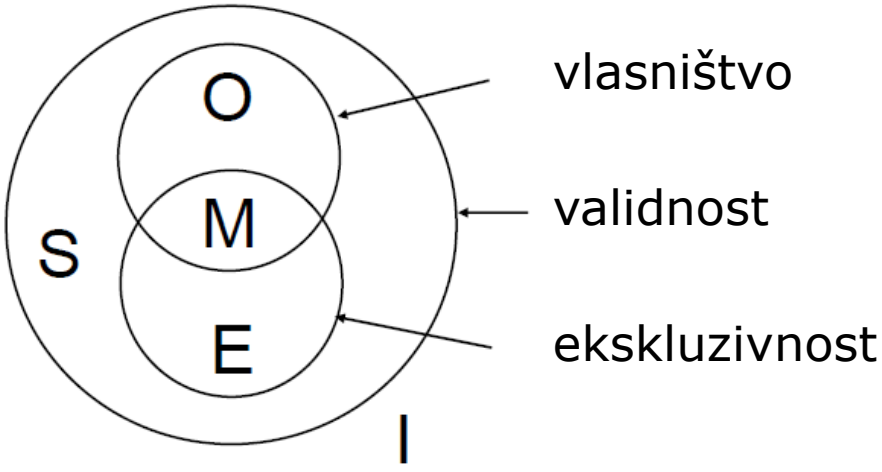
U ovim aplikacijama retko E -> M

MESI Protokol – poboljšanja

- Direktan prenos C -> C
  - Prihvatanje iz keš memorije brže
  - Štedi propusni opseg memorije
  - Veća HW složenost
- Problemi
  - Memorija mora da čeka da li će se keš memorija javiti
  - Potrebna arbitracija kada ima više kopija
- MESIF
  - Dodatno stanje F(Forward)
  - Keš memorija sa blokom u stanju F opslužuje promašaj
  - Koristi se u Intel-ovim procesorima

MESI Protokol – poboljšanja

- Ažuriranje memorije pri prelazu iz M u S
  - Zbog mogućnosti da se izgubi pri zameni bloka u S
  - ... ali nepotrebno troši propusni opseg ako se opet upisuje (ponekad često u *producer-consumer*)
- Rešenje - MOESI
  - Owned (O) = Shared + Dirty
  - Odgovornost za WB, memorija neažurna
  - Može više kopija (jedna O, ostale S)
  - U njega se prelazi iz M izbegavajući WB
  - Nadskup protokola
  - AMD Opteron



Ažurirajući protokoli

- Invalidacioni protokoli neefikasni za
  - Kraće upisne nizove
  - Finiju granularnost deljivosti
- Primer- sinhronizacija preko deljenog flaga
  - p0 čeka da bude 0, onda radi i postavi ga na 1
  - p1 čeka da bude 1, onda radi i postavi ga na 0
  - Broj potrebnih transakcija?
- Strategija ažuriranja (*update*)
  - Distribuirani upis ažurira sve kopije deljenog podatka
  - Ažuriranje bez zahteva (*non-demand*)
  - Jeftina transakcija na magistrali
  - WT za deljene, WB za privatne podatke
  - MRMW

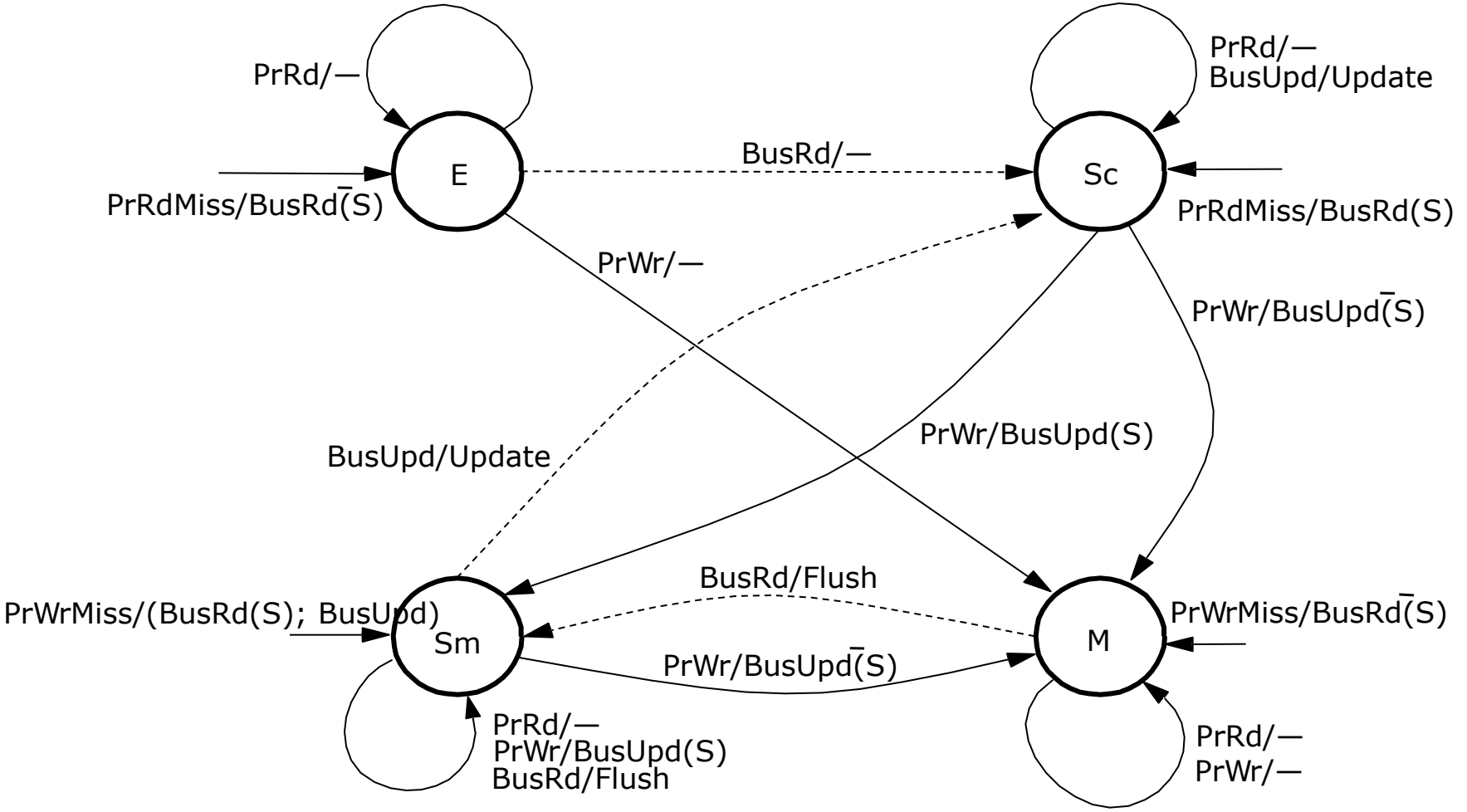
Dragon protokol

- Xerox PARC Dragon radna stanica
- Stanja (nema nevažećeg stanja – I !) :
  - Exclusive-clean (E) – jedina kopija, memorija ažurna
  - Shared-clean (Sc) – moguće dve ili više kopija, memorija ažurna ili neažurna
  - Shared-modified (Sm) – moguće više kopija (ostale Sc), memorija neažurna,vlasnik, odgovornost za WB
  - Modified (M) – jedina kopija, memorija neažurna, vlasnik, odgovornost za WB
- Transakcije na magistrali
  - Čitanje (BusRd) i write-back (Flush) na nivou bloka
  - Distribuirani upis (BusUpd) na nivou reči (ne za memoriju!)
  - Dinamička detekcija deljivosti – signal S

Dragon protokol – akcije protokola

- RM - BusRd
  - Podaci dolaze od Sm ili M (-> Sm), stanje -> Sc (S)
  - ... ili od memorije, stanje -> Sc (S) ili E (S/)
- WH
  - U stanju M ili E (-> M) lokalni upis (bez izlaska na magistralu)
  - U stanju Sm ili Sc izdaje se BusUpd, stanje -> Sm (S) ili M (S/)
- WM = RM + WH
  - Izdaje se BusRd, ako je S/ ide u M, inače BusUpd i ide u Sm
- Zamena
  - Ako je u Sm ili M -> *write-back*

Dragon protokol – dijagram stanja



Dragon protokol – projektne odluke

- Stanje Sm neophodno?
  - Nije, ako BusUpd ažurira memoriju (npr., DEC Firefly)
  - Uvedeno u Dragonu zbog razlike brzina SRAM-a i DRAM-a
- Objavljivanje izbacivanja Sc kopije?
  - Mogla bi se prepoznati ekskluzivnost (Sc -> E ili Sm -> M)
  - Zamena nije na kritičnom putu, eventualno kasnije ažuriranje jeste
  - Povećava složenost
- Korektnost koherencije i konzistencije
  - Svi upisi se "vide" na magistrali kao kod WTI
  - Za upise garantovano:
    - serijalizacija
    - detektovanje kraja
    - atomičnost

Evaluacija projektnih odluka

- Projektne odluke u MP složene
- Projektne odluke u CCP, takođe, složene
  - U interakciji sa drugim sistemskim odlukama
  - Utiču na latenciju pristupa i propusni opseg
  - Zasnovane na realističnoj evaluaciji
  - ... ali i na iskustvu, intuiciji, ...
- Zahtevi realistične evaluacije
  - Složeni simulatori (npr., Simics, SimOS, M5, RSIM, ...)
  - Reprezentativna radna opterećenja (npr., SPLASH-2)
  - Odgovarajuća metrika performansi (vreme izvršavanja, snaga sistema, propusni opseg, statistika događaja, ...)
  - Odgovarajuće skaliranje problema

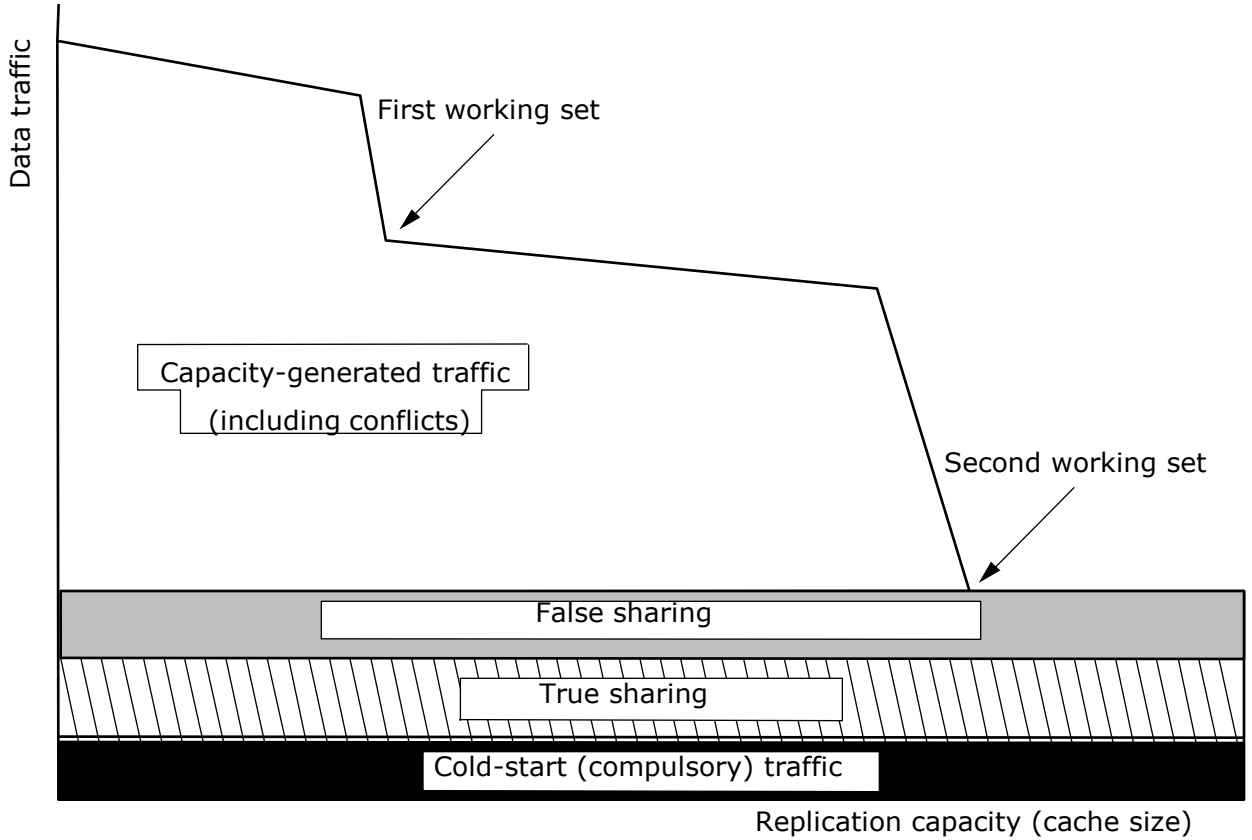
Promašaji u keš memorijama

- Model 3C u jednoprocesorskim sistemima
  - *Compulsory* - prvi pristup, obavezni (*cold*)
  - *Capacity* - usled ograničene veličine (čak i u FA)
  - *Conflict* - usled ograničene set-asocijativnosti
- Broj promašaja se smanjuje
  - Povećanjem veličine keš memorije
  - Povećanjem veličine bloka
  - Povećanjem set asocijativnosti
- U multiprocesorskim sistemima nova vrsta – promašaji usled deljenja podataka (*coherence*)
  - Pravo deljenje (*true sharing*)
  - Lažno deljenje (*false sharing*)
- Postoje i kombinovani promašaji

Promašaji usled deljenja podataka

- Pravo deljenje
  - Reč koju upiše jedan procesor kasnije koristi drugi
  - Prava komunikacija neophodna za korektno izvršavanje
  - Promašaj donosi potrebni novi podatak
  - Inherentni paralelizaciji
  - Postojali bi i u kešu beskonačne veličine
  - Čine značajan deo ukupnog broja promašaja (zavisno od aplikacije)
- Lažno deljenje
  - Različite reči kojima pristupaju različiti procesori (bar jedan upisuje!), a pripadaju istom bloku
  - Promašaj ne donosi novi podatak
  - Ne javlja se ako je jedinica koherencije - reč

Uticaj veličine keš memorije

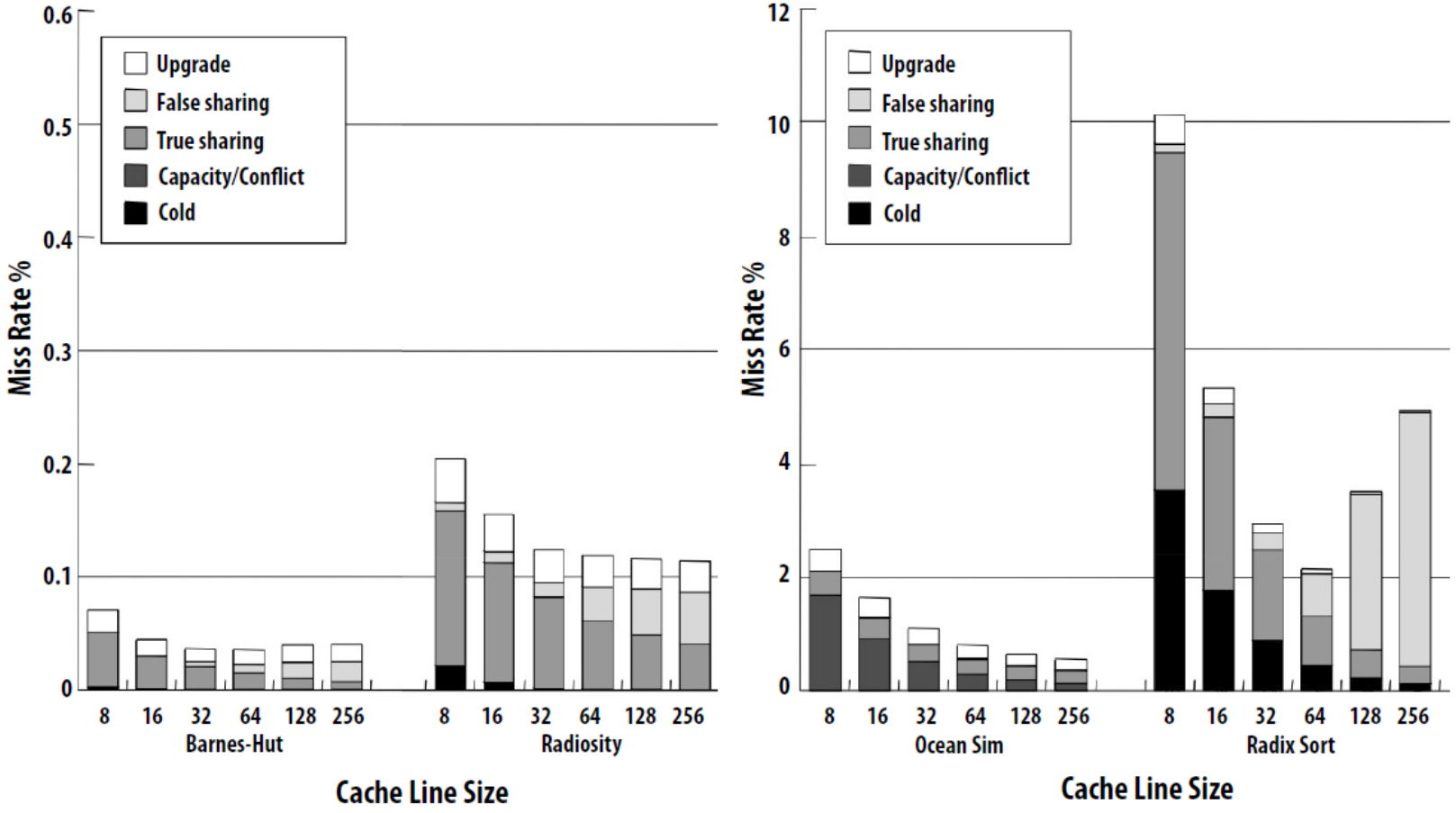


- Model radnog skupa zasnovan na vremenskoj lokalnosti
  - Skup podataka koji se pretežno koristi u nekom periodu

Uticaj veličine bloka

- Povećanje veličine bloka
  - Tehnološki trend u prevazilaženju memorijskog "*gap*"- a
- Pozitivni efekti
  - Smanjuje broj obaveznih promašaja
  - Koristi prostornu lokalnost (efekt "*prefetching*"-a)
  - Smanjuje i broj promašaja usled prave deljivosti
  - Amortizuje fiksni "*overhead*" transakcije i pristupa memoriji
- Negativni efekti
  - Povećava broj promašaja usled lažne deljivosti
  - Dohvatanje nepotrebnih podataka (prostorna lokalnost u paralelnim programima manja nego u sekvencijalnim)
  - Povećava broj konfliktnih promašaja
  - Povećavaju cenu promašaja (može se ublažiti!)
  - Može da se povećava saobraćaj na magistrali i kontencija

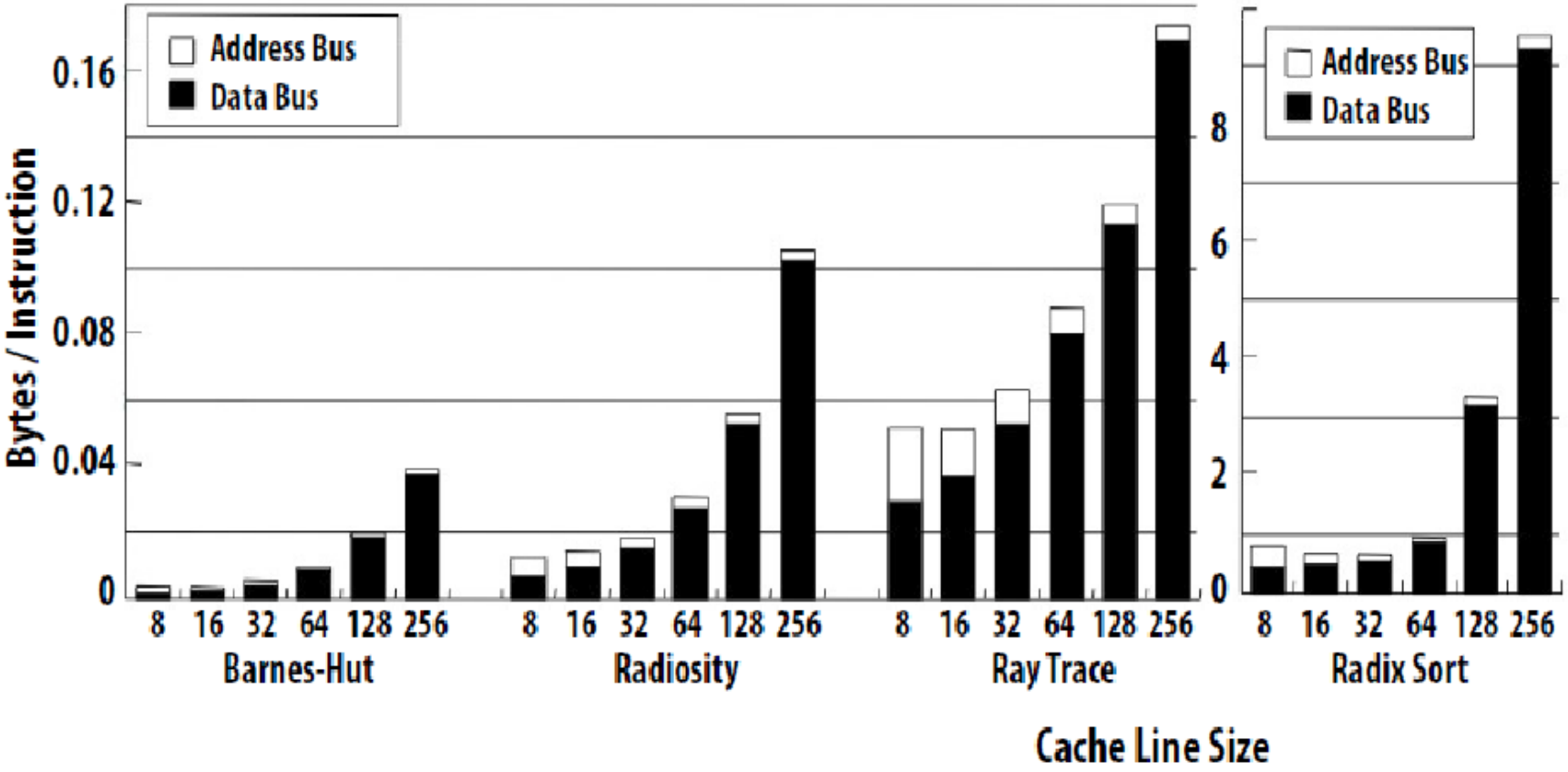
Uticaj veličine bloka



1 MB *4-way* $M

Upgrade – upis u S (zaustavljanje+saobraćaj)

Uticaj veličine bloka



Uticaj veličine bloka

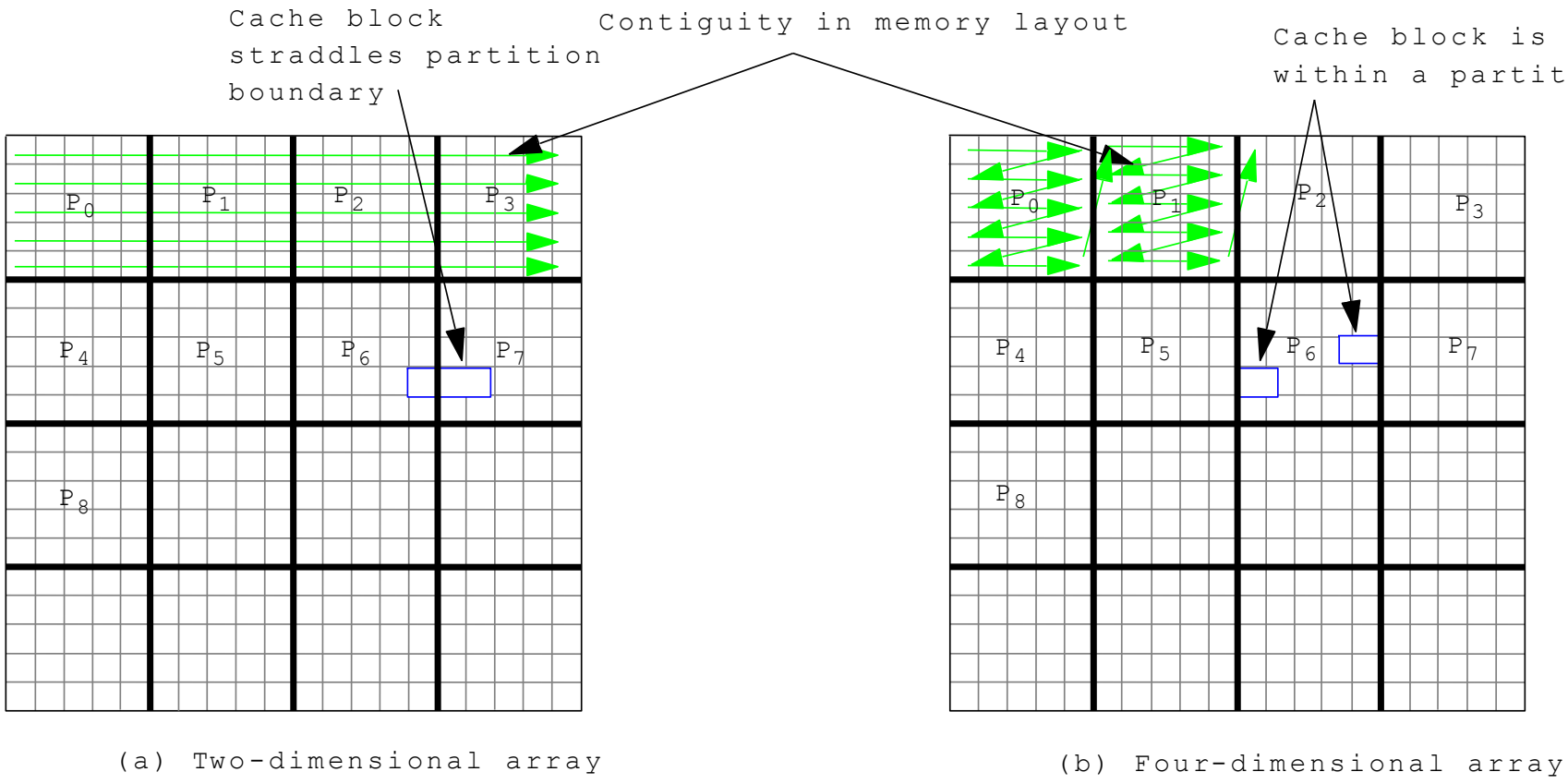
- Ublažavanje negativnih efekata
  - U sve većim L1 i L2 povećanje broja konfliktnih promašaja srazmerno malo
  - Sektorsko keširanje
  - Promenljiva veličina bloka (složeno!)
  - SW tehnike za smanjenje lažne deljivosti i poboljšanja lokalnosti
  - Strategija ažuriranja

- Sektorsko keširanje
  - Jedinica adresiranja i prenosa - blok
  - Jedinica koherencije – podblok
  - Posebni bitovi stanja na nivou podbloka

Implikacije za SW

- Izbegavanje migracije procesa
- Izbegavanje deljenja sa upisom (*write sharing*) ili ga, bar, vremenski lokalizovati
  - Često zahteva sinhronizaciju (još skuplje!)
- Smanjiti prostorno preplitanje pristupa
  - Pri raspoređivanju posla procesorima (npr. obrada niza)
  - Pri strukturiranju podataka (npr., 4D umesto 2D da bi se obezbedila kontinualnost particije)
- Izbegavati konfliktne promašaje
  - Iako je logička veličina 2^i, alocirati veću fizičku veličinu
- Kopirati razdvojene podatke pre korišćenja
  - Privremene kontinualne strukture (cena kopiranja!)

Implikacije za SW



Implikacije za SW

- Koristiti odvojene "*heap*"-ove za procese
- Organizacija niza zapisa
- Vezati niz za početak bloka (*alignment*)
- Dopunjavati elemente niza na veličinu bloka (*padding*)

- Potencijalni problem!

```
int myCounter[NUM_THREADS];
```

Rešenje

```
struct PerThreadState {
  int myCounter;
  char padding[64 - sizeof(int)];
};
PerThreadState myCounter[NUM_THREADS];
```

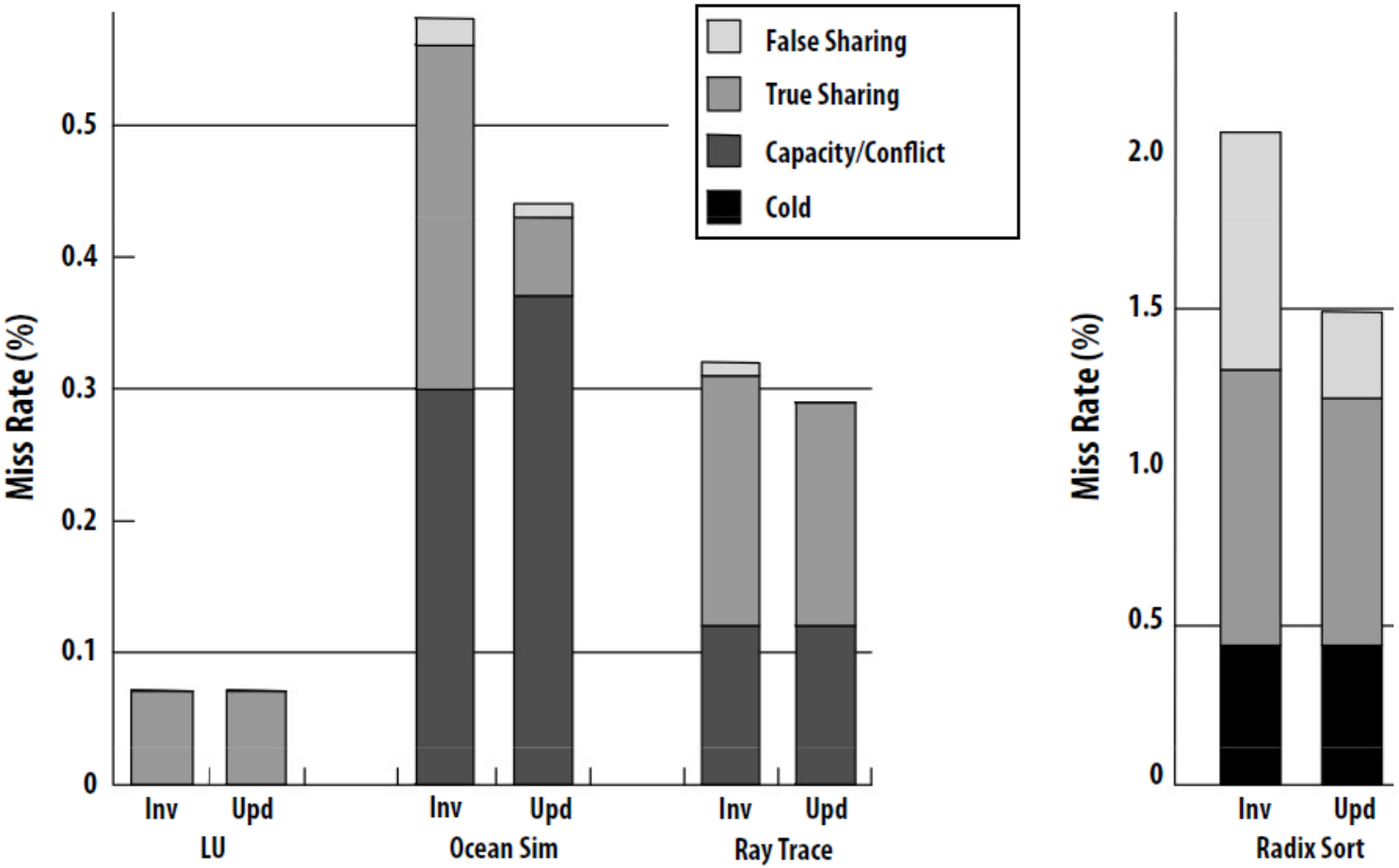
Poništavanje ili ažuriranje

- Izbor zavisi od karakteristika aplikacije
- Osnovni kriterijum – procesorska lokalnost
  - Indikator - dužina upisnog niza (*write run*)
- Poništavanje
  - "Čisti" keš memoriju od zaostalih kopija koje se ne koriste
  - Cena: invalidacioni promašaji (može i "*read snarfing*")
  - Lokalni, jeftini upisi
  - Dobro za duže upisne nizove
  - Loše za kraće upisne nizove i deljenje tipa 1 *proizvođač – "mnogo korisnika"*
- Ažuriranje
  - Cena operacije, obično, ista kao i za invalidaciju
  - Bolje za kraće upisne nizove
  - Lošije za duže upisne nizove i migraciju procesa

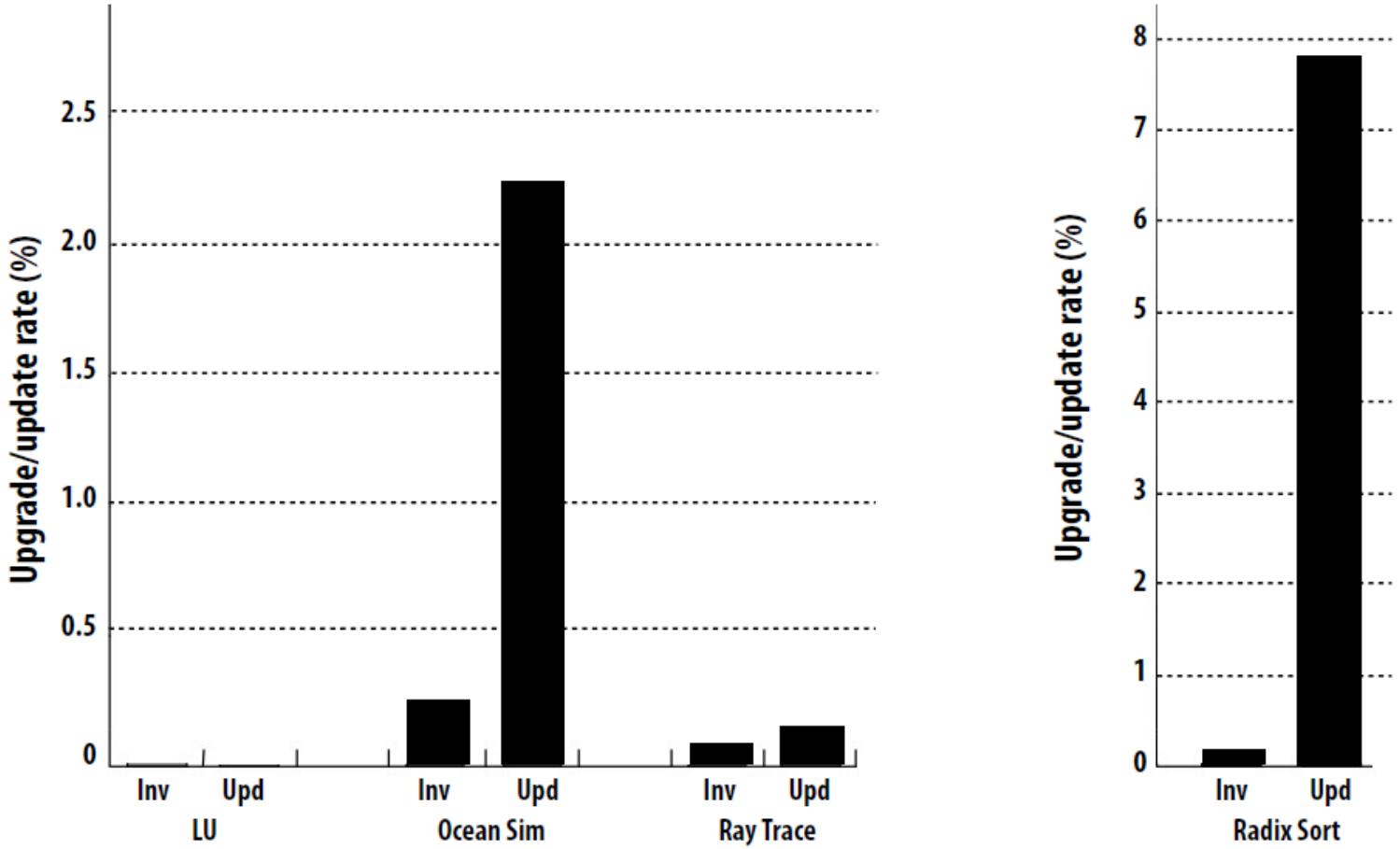
Izbor strategije upisa - primer

- SP1: {P1: Write V; P2..n: Read V} x k // 1-n
- SP2: {(P1: Write V) x m; P2: Read V} x k // 1-1
- Pretpostavke:
  - n = 16, m = 10, k = 10
  - inv/upg = 6 bajta (5 adr + 1 cmd)
  - upd = 14 bajta (6 + 8)
  - miss = 70 bajta (6 + 64)
- WU
  - SP1: n x RdMiss + (k-1)x upd = 1260 bajta
  - SP2: 2 x RdMiss + m x (k-1)x upd = 1400 bajta
- WI
  - SP1: (n + (k-1)x(n-1)) x RdMiss + (k-1)x upg = 10624 bajta
  - SP2: (2 + k-1)x RdMiss + (k-1)x upg = 824 bajta

Poništavanje ili ažuriranje



Poništavanje ili ažuriranje



Savremeni Intel i AMD procesori koriste invalidaciju!

Adaptivni protokoli

- Izbor WI ili WU na nivou stranice uz SW podršku
  - Podrška kroz TLB
  - Odluka preko sistemskog poziva
  - Previše krupna granularnost
  - Nameće obavezu programeru
  - Primer - MIPS R4000

- Izbor WI ili WU na nivou bloka uz HW podršku
  - Dinamička detekcija načina deljivosti određuje odgovarajuću strategiju upisa
  - Transparentno za korisnika
  - Obično počnu sa WU, pa ako treba prebace na WI
  - Hardverska složenost

Adaptivni protokoli - primeri

- RWB (*Read & Write Broadcast*)
  - Rudolph, Segall - CMU
  - Prvi upis WU, a drugi WI
  - Novo stanje i transakcija
  - Prerana invalidacija (za potrebe sinhronizacije)

- EDWP (*Efficient Distributed Write Protocol*)
  - Archibald - UoW
  - Invalidacioni prag – 3 upisa (na osnovu rezultata simulacije)
  - Tri dodatna stanja – Sco, Rw1 i Rw2
  - Nova linija na magistrali D (da li postoji vlasnik u stanju M)
  - "Čisti" vlasnik – poslednji keš koji je imao promašaj za blok
  - Sličan protokol – UpdateOnce (invalidacija posle RW1)

Adaptivni protokoli - EDWP

Operation	C1	C2	C3	C4	S	D
Initial state	-	Sc	Sc	Sco	-	-
P1: Read X	Sco	Sc	Sc	Sc	1	0
P1: Write X	Sm	Rw1	Rw1	Rw1	1	-
P3: Read X	Sm	Rw1	Sc	Rw1	-	-
P1: Write X	Sm	Rw2	Rw1	Rw2	1	-
P1: Write X	Sm	Rw2	Rw2	Rw2	1	-
P1: Write X	M	Inv	Inv	Inv	0	-
P3: Read X	Sm	Inv	Sc	Inv	1	1

Implementacija

- Ciljevi (ponekad kontradiktorni)
  - Korektnost
  - Performanse
  - Što manja HW složenost
- Korektnost
  - *Deadlock* (potpuno blokiranje,
    često u *request-reply* protokolima,
    zahtevi ne smeju odlagati odgovore)
  - *Livelock* (nema napretka, iako se transakcije odvijaju,
    npr., simultani upisi u deljeni blok,
    ne razdvajati dobijanje vlasništva i upis)
  - *Starvation* (neki su zapostavljeni u korišćenju resursa,
    rešava se poštenom arbitracijom i FIFO redovima)

Implementacija

- Vrlo složeni elementi implementacije
  - Magistrala sa razdvojenim, simultanim transakcijama
  - Implementacija RMW
  - Serijalizacija i propagacija upisa
  - Kolektivni odgovori na magistrali (*wired-OR*)
  - Dvostruki tagovi, ako ima samo L1
  - Prioritet se daje promašajima,
    upisni baferi za odlaganje *write-backa* (ali mora *snoop*!)
  - Neatomski prelazi (više operacija zahteva međustanja)

Implementacija