Multiprocesorski sistemi

Osluškujući (*snoopy)* protokoli

Marko Mišić, Milo Tomašević

13S114MUPS, 13E114MUPS, 13M114MUPS

2024/2025.

"*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)

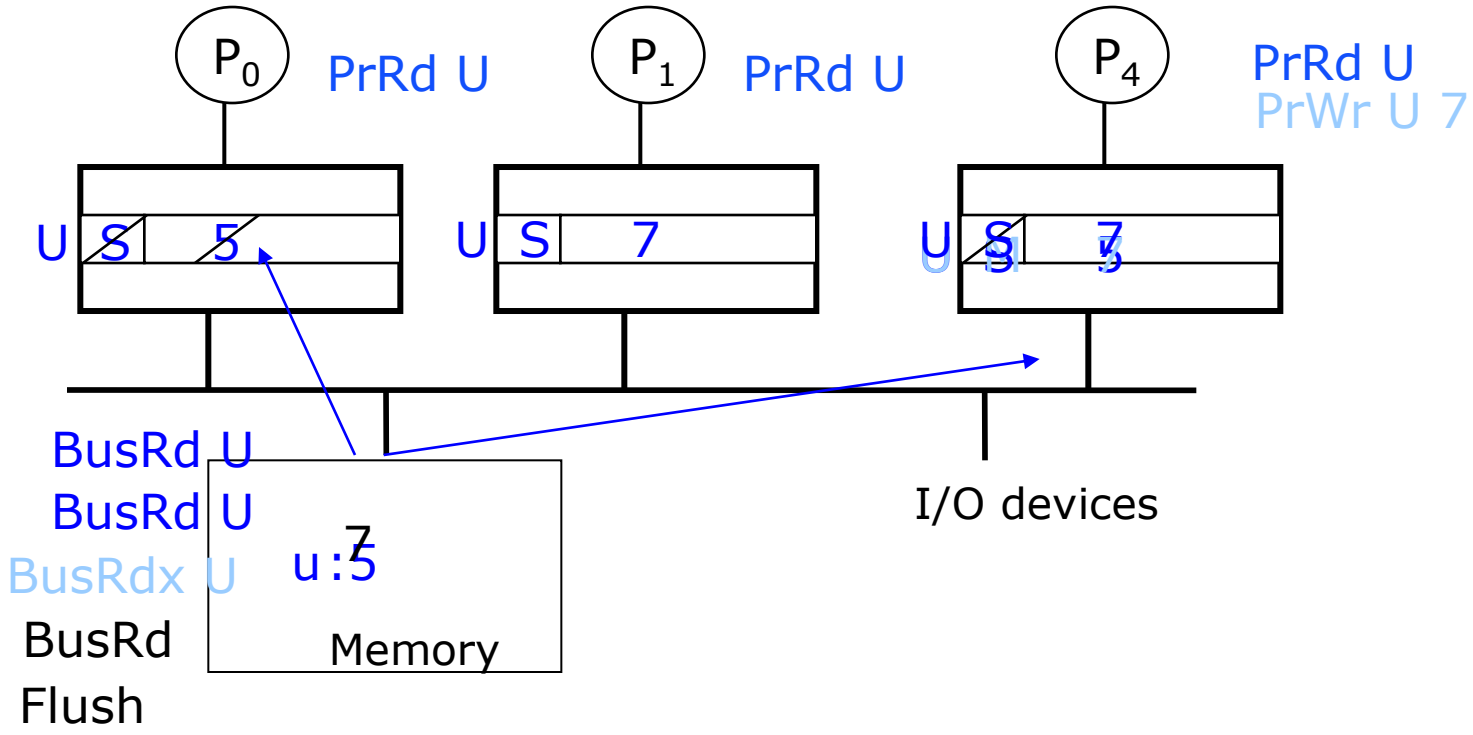
Trenutni ili odloženi upis

- 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

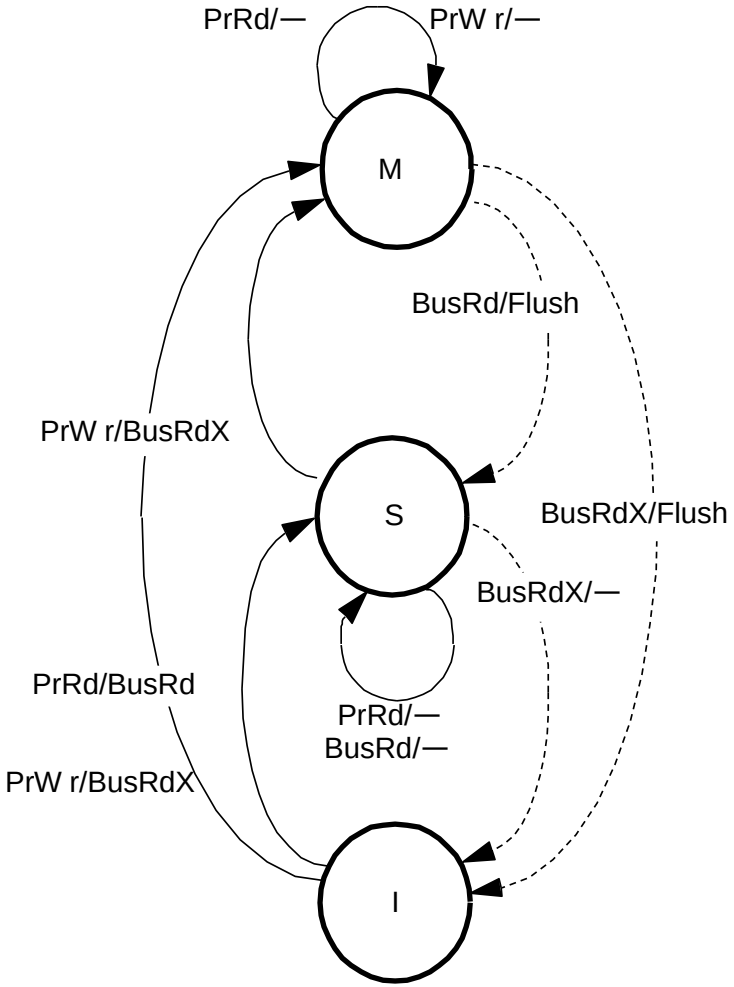
- Stanja
  - Invalid (I) – nevažeći ili blok nije prisutan
  - Shared (S) – važeća kopija, memorija ažurna, može, ali ne mora biti drugih kopija
  - Modified (M) – jedina važeća kopija, memorija neažurna, vlasništvo, ekskluzivnost
- Operacije procesora
  - PrRd – čitanje
  - PrWr - upis
- Transakcije 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

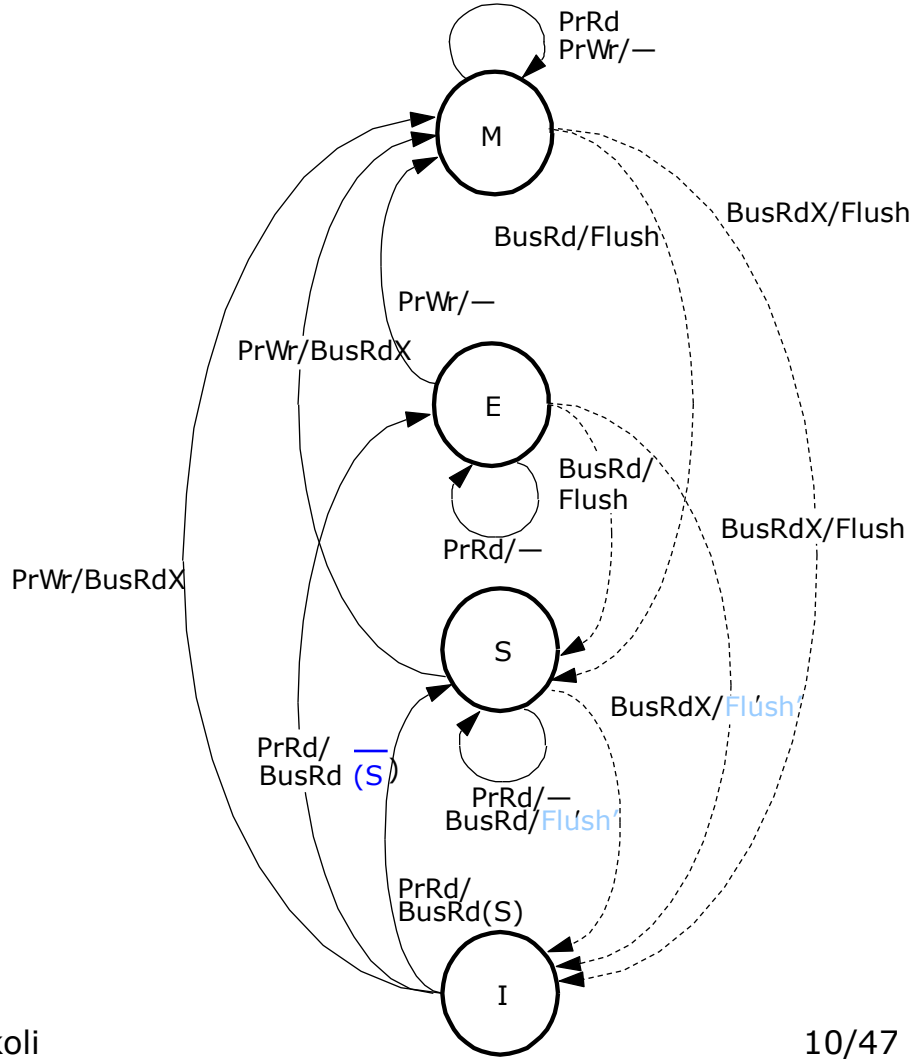
- Šta uraditi na BusRd u stanju M ?
  - Ako se očekuje dalje čitanje, M -> S (*read sharing*)
  - Ako se očekuje migratorna deljivost, M -> I
- Problem – invalidacioni promašaji
  - Rešenje – validacija bloka
  - *Read Broadcast* (*Read Snarfing*)
  - Samo jedan invalidacioni promašaj po bloku
  - Povećana 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

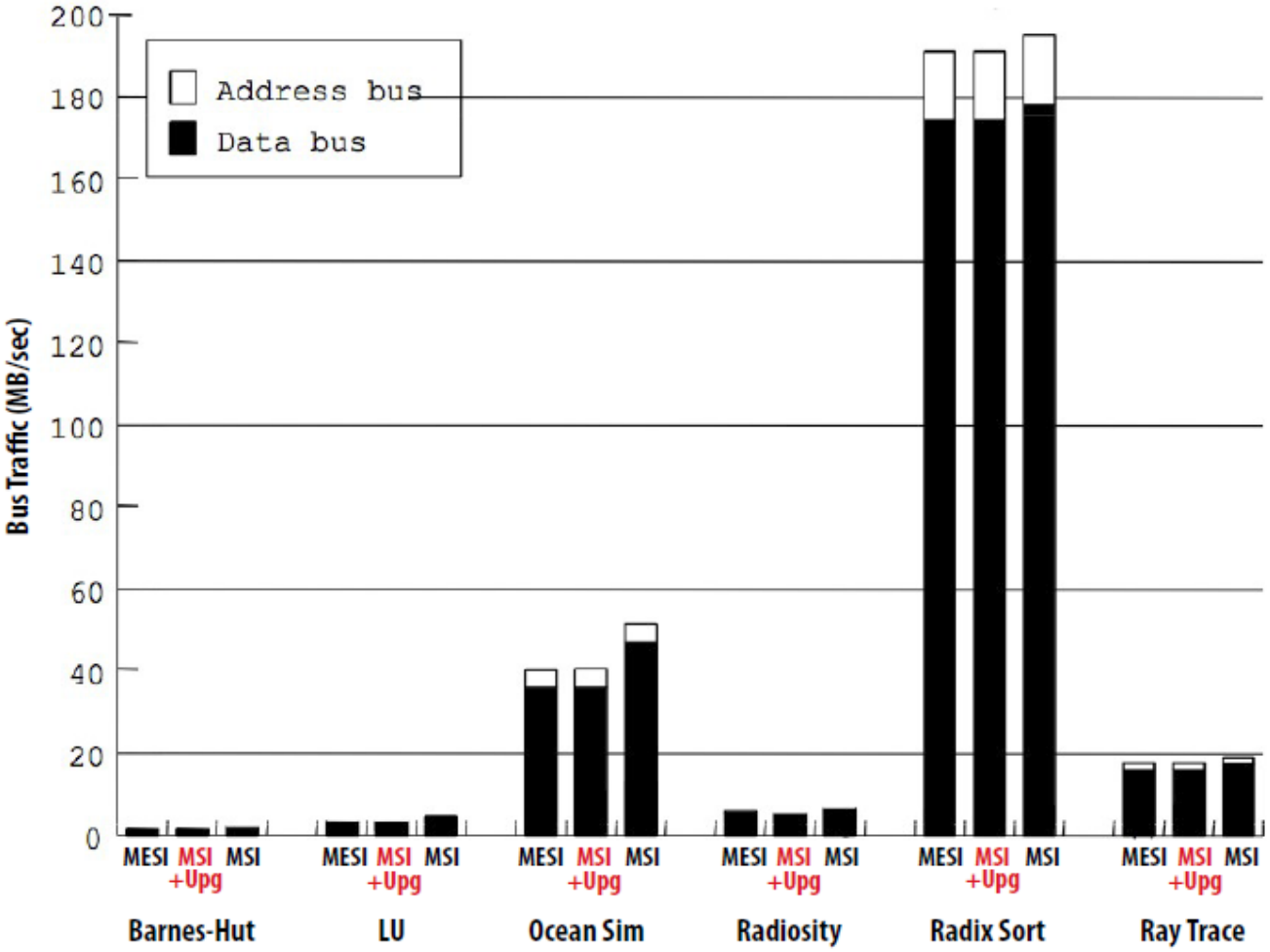
- Nadgradnja MSI protokola
  - Exclusive (E) – jedina važeća kopija, memorija ažurna
  - Upis u ovom stanju ne zahteva izlaz na magistralu, već samo ažuriranje stanja
  - Stanje S sada, praktično, označava pravo deljenje
  - Illinois (Papamarcos, Patel)
  - Korišćen u mnogim komercijalnim procesorima
- 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 u smislu transakcija
- Obična zamena bloka može napraviti nelogično stanje
  - Može ostaviti u stanju S samo jednu kopiju



MSI vs. MESI



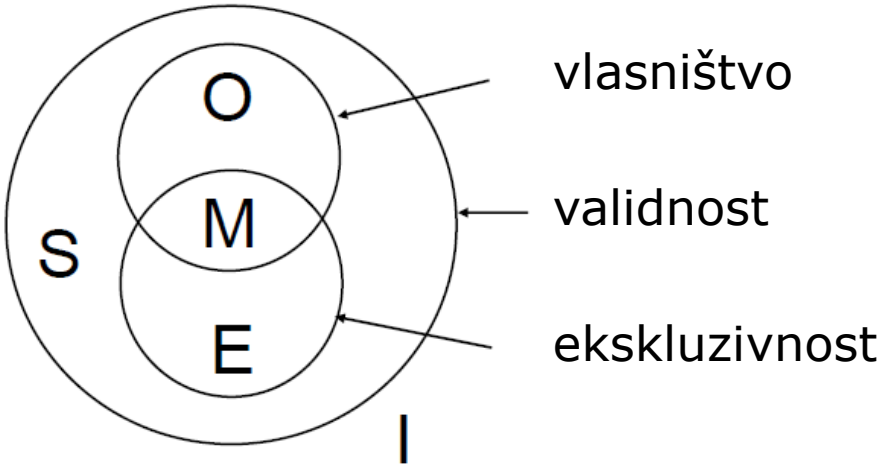
U ovim aplikacijama retko E -> M

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)
    - Onaj ko je poslednji tražio podatak
  - Keš memorija sa blokom u stanju F opslužuje promašaj
  - Koristi se u Intel-ovim procesorima

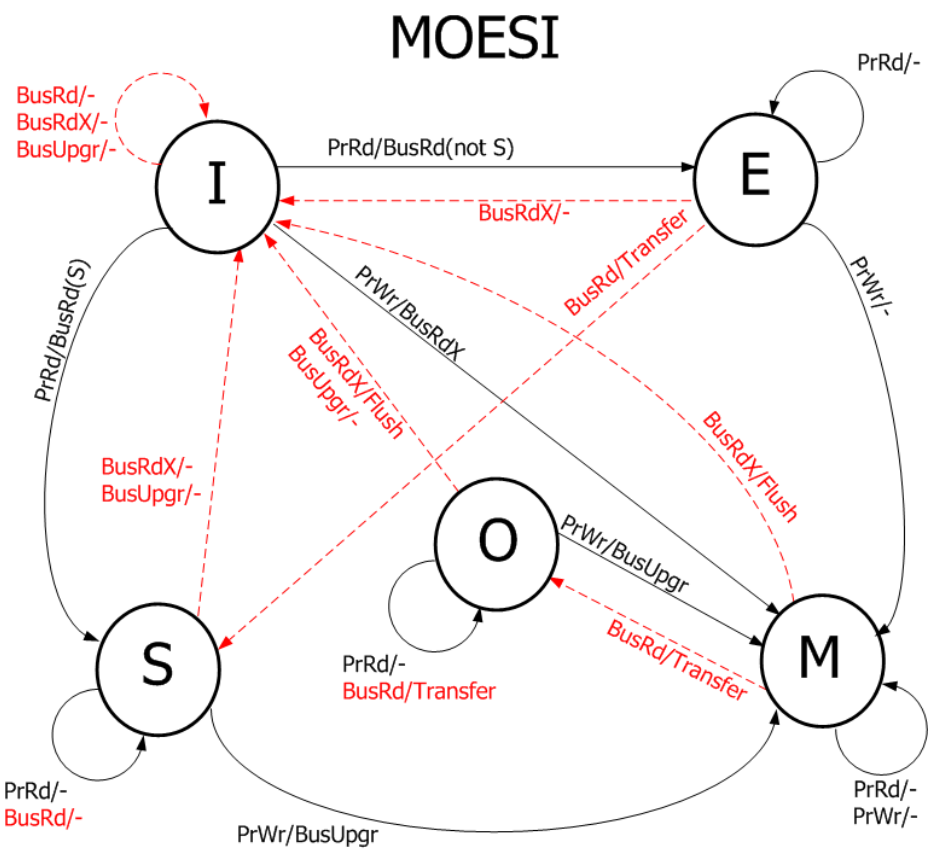
MESI Protokol – poboljšanja

- 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



MOESI protokol

- RM povlači iz drugog keša
  - U stanju M, E ili O
  - Nema ažuriranja memorije
- Ažuriranje memorije tek kod zamene
  - Blok u stanju O ili M



BusUpgr – Invalidira ostale kopije
Transfer – prosleđuje blok drugom procesoru, bez ažuriranja memorije
Flush – ažurira memoriju i (opciono) prosleđuje blok drugom procesoru

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

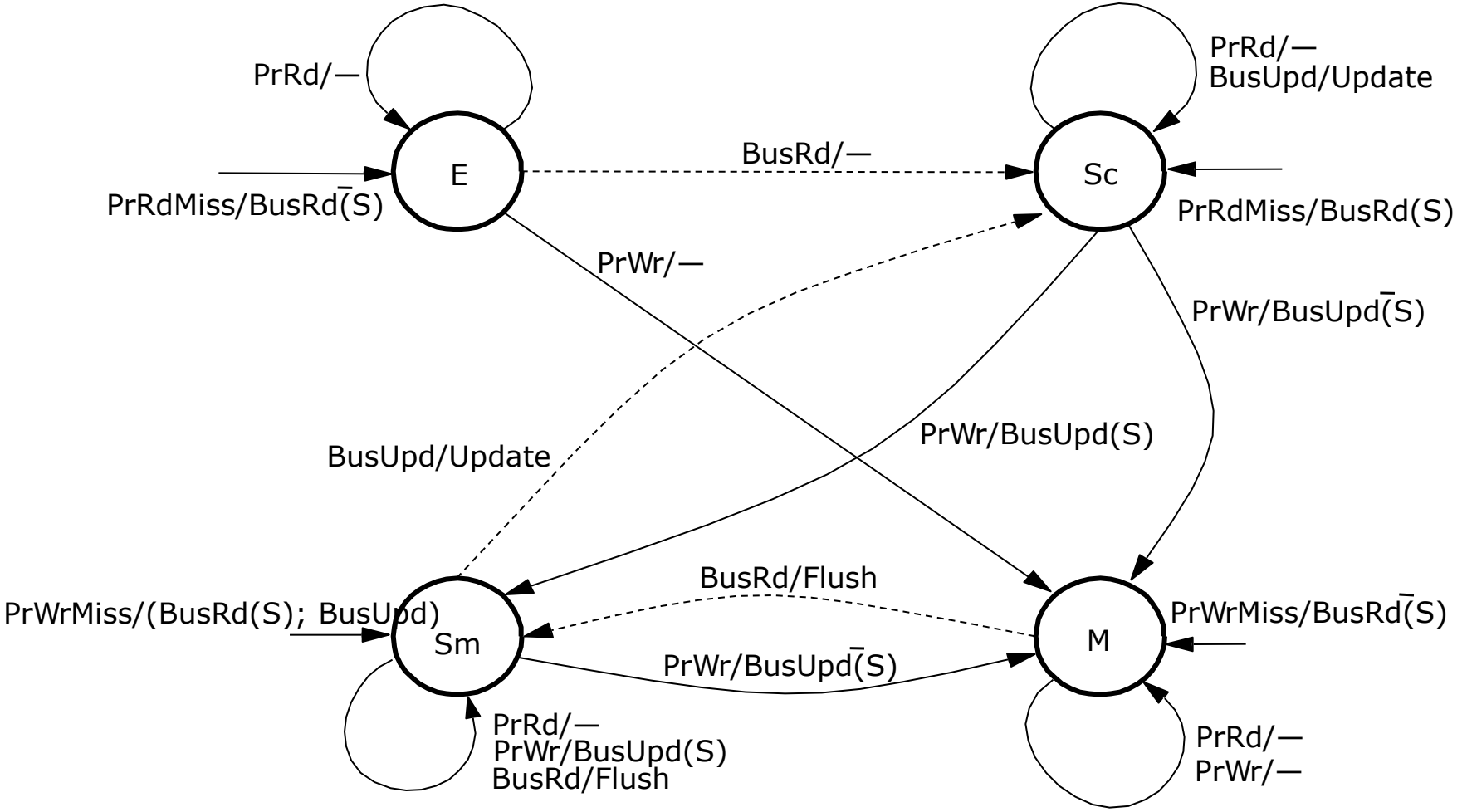
Dragon protokol

- 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

- RH – bez promene stanja i akcije na magistrali
- RM - BusRd
  - Podaci dolaze od Sm ili M (-> Sm), stanje -> Sc (S)
  - ... ili od memorije, stanje -> Sc (S) ili E (not 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 (not S)
- WM = RM + WH
  - Izdaje se BusRd, ako je (not S) ide u M, inače BusUpd i ide u Sm
- Zamena
  - Ako je u Sm ili M -> *write-back*

Dragon protokol – dijagram stanja

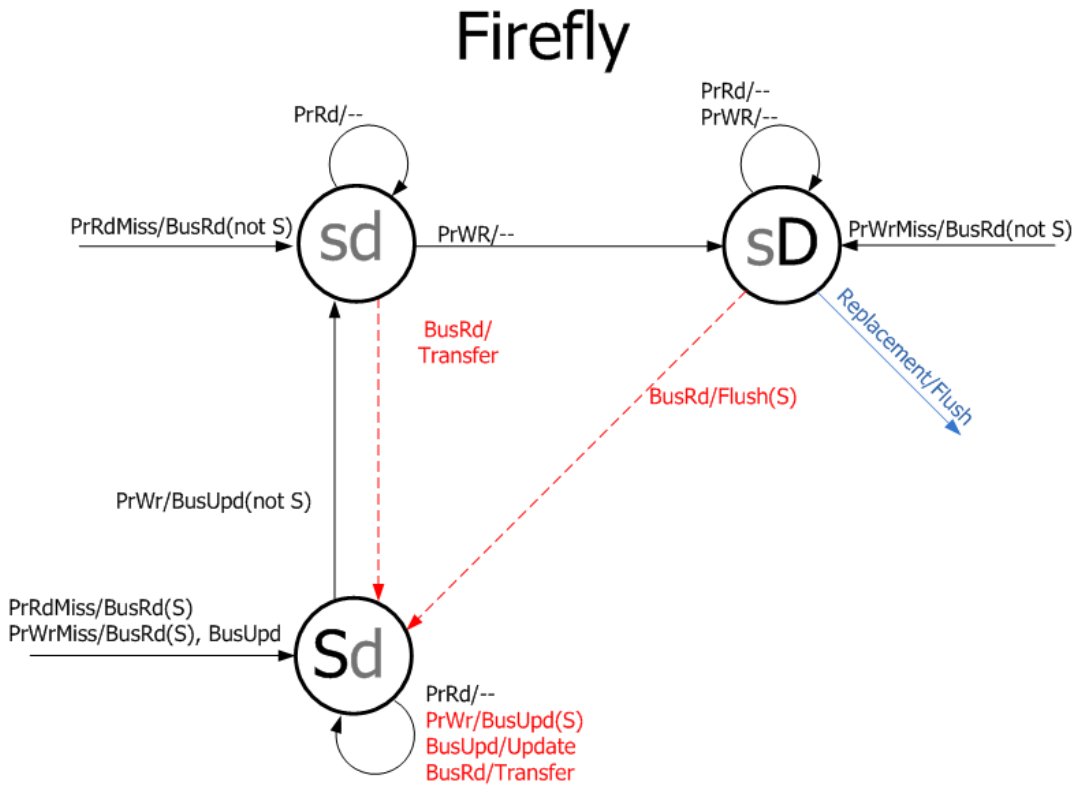


Dragon protokol – projektne odluke

- 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

Firefly protokol

- Eliminacija Sm stanja
  - Kombinacija
  - s/S – not/shared
  - d/D not/dirty
- BusUpd ažurira podataka i u memoriji



Flush – ažurira blok u memoriji i eventualno dostavlja procesoru koji je zahtevao podatak
BusUpd – ažurira samo promenjeni podatak u svim kopijama i u memoriji
Transfer – prenos celog bloka drugom procesoru iz keširane kopije

Napomena: magistrala mora biti realizovana tako da korektno radi ukoliko više procesora sa kopijom u stanju Sd uradi operaciju **Transfer**. Ukoliko to tehnološki nije moguće, mora se raditi čitanje iz glavne memorije, a ne iz drugog keša.

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
    - Simics, gem5, ranije 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 (1)

- Promašaji su ono što košta!
- Model 3C u jednoprocesorskim sistemima
  - *Compulsory* - prvi pristup, obavezni (*cold*)
    - Podatak nije nikad bio u kešu
  - *Capacity* - usled ograničene veličine
    - Podatak je bio u kešu,
      ali je zamenjen usled nedostatka prostora
    - Dešava se čak i u *full-associative* keš memorijama,
      kada je radni skup veći
  - *Conflict* - usled ograničene set-asocijativnosti
    - Podatak je bio u kešu, ali je zamenjen,
      jer nivo asocijativnosti nije bio dovoljan

Promašaji u keš memorijama (2)

- Broj promašaja se smanjuje
  - Povećanjem veličine keš memorije (*capacity*)
    - Radni skup može da stane u keš memoriju
  - Povećanjem veličine bloka (*compulsory*)
    - Manji broj odlazaka u memoriju
  - Povećanjem set asocijativnosti (*conflict*)
    - Manja konkurencija za mesto u jednom setu
- U multiprocesorskim sistemima nova vrsta – promašaji usled deljenja podataka (*coherence*)
  - Pravo deljenje (*true sharing*)
  - Lažno deljenje (*false sharing*)
  - Promašaj indukovan događajem na magistrali
  - Model **4C**
- Postoje i kombinovani promašaji

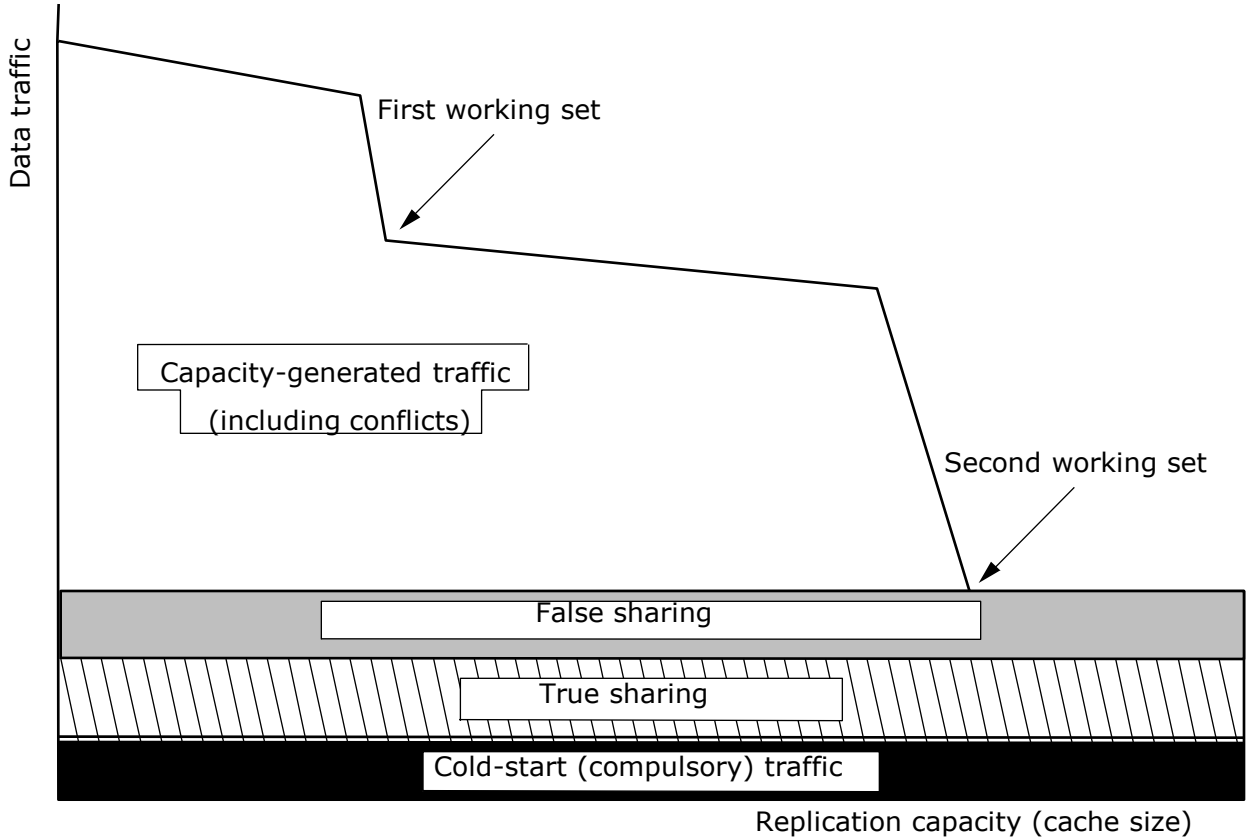
Promašaji usled deljenja podataka (1)

- Pravo deljenje
  - Reč koju upiše jedan procesor kasnije koristi drugi
  - Invalidacionim promašajem prosleđuje se ažurna reč
  - Prava komunikacija neophodna za korektno izvršavanje
- Promašaj donosi potrebni novi podatak
  - Šablon komunikacije inherentni paralelizaciji
  - Postojali bi i u kešu beskonačne veličine
    - Posledica strategije održavanja koherencije
- Čine značajan deo ukupnog broja promašaja
  - Zavisno od aplikacije

Promašaji usled deljenja podataka (2)

- Lažno deljenje - fenomen sa negativnim posledicama
  - Mogućnost da logički privatni podaci različitih procesora budu u istom bloku
    - Različite reči kojima pristupaju različiti procesori (bar jedan upisuje!), a pripadaju istom bloku
- Primer sa blokom od dve reči
  - Procesor P0 čita i piše reč A
  - Procesor P1 čita i piše reč B
    - Invalidacija bloka se radi na svaki upis
    - *Ping-pong* efekat
  - Promašaj ne donosi novi podatak
- Strategija razrešavanja
  - Ne javlja se ako je jedinica koherencije – reč
  - Dopunjavanje (*padding*) podataka, reorganizacija struktura podataka

Uticaj veličine keš memorije

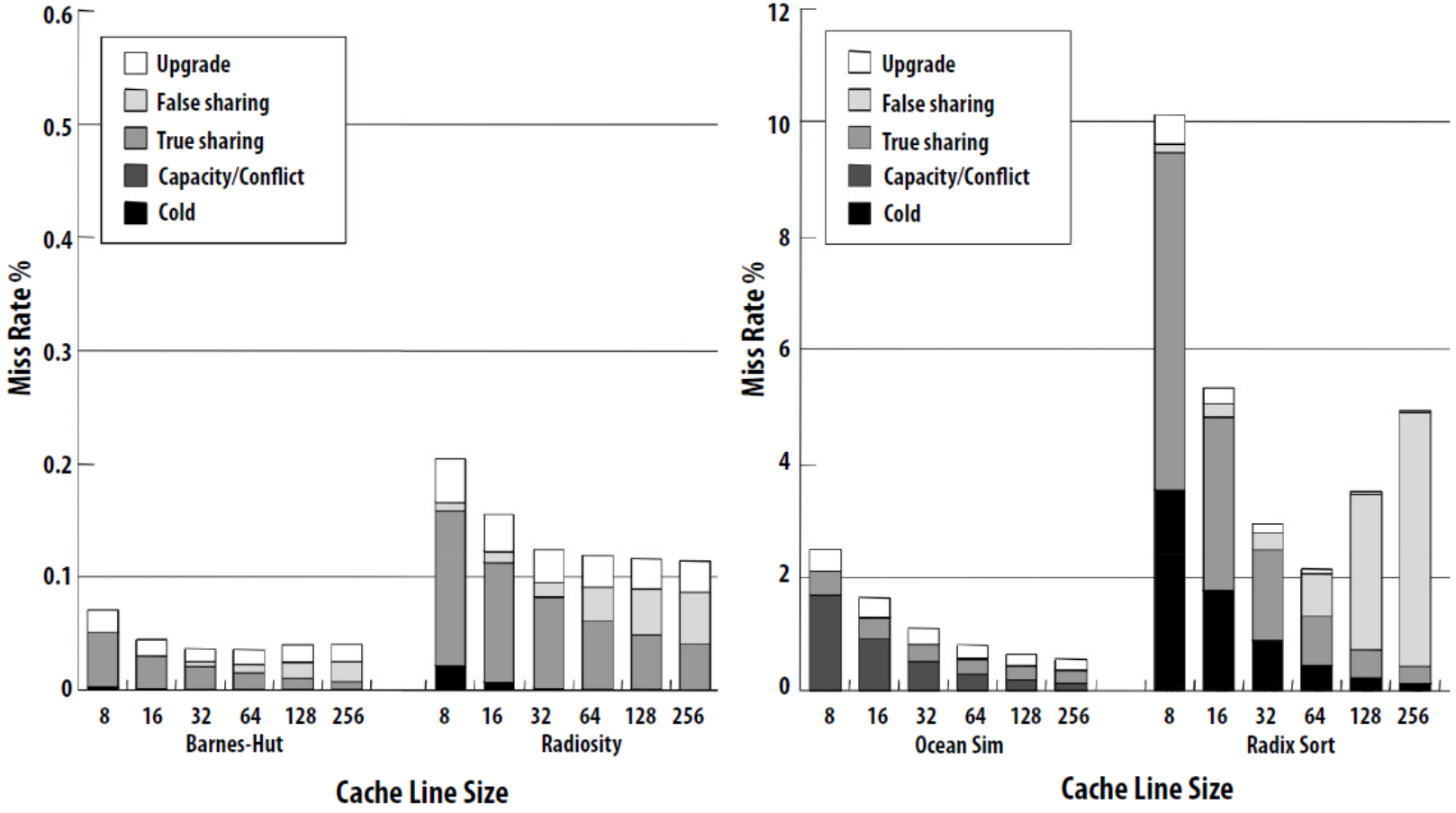


- Model radnog skupa zasnovan na vremenskoj lokalnosti
  - Skup podataka koji se pretežno koristi u nekom periodu

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

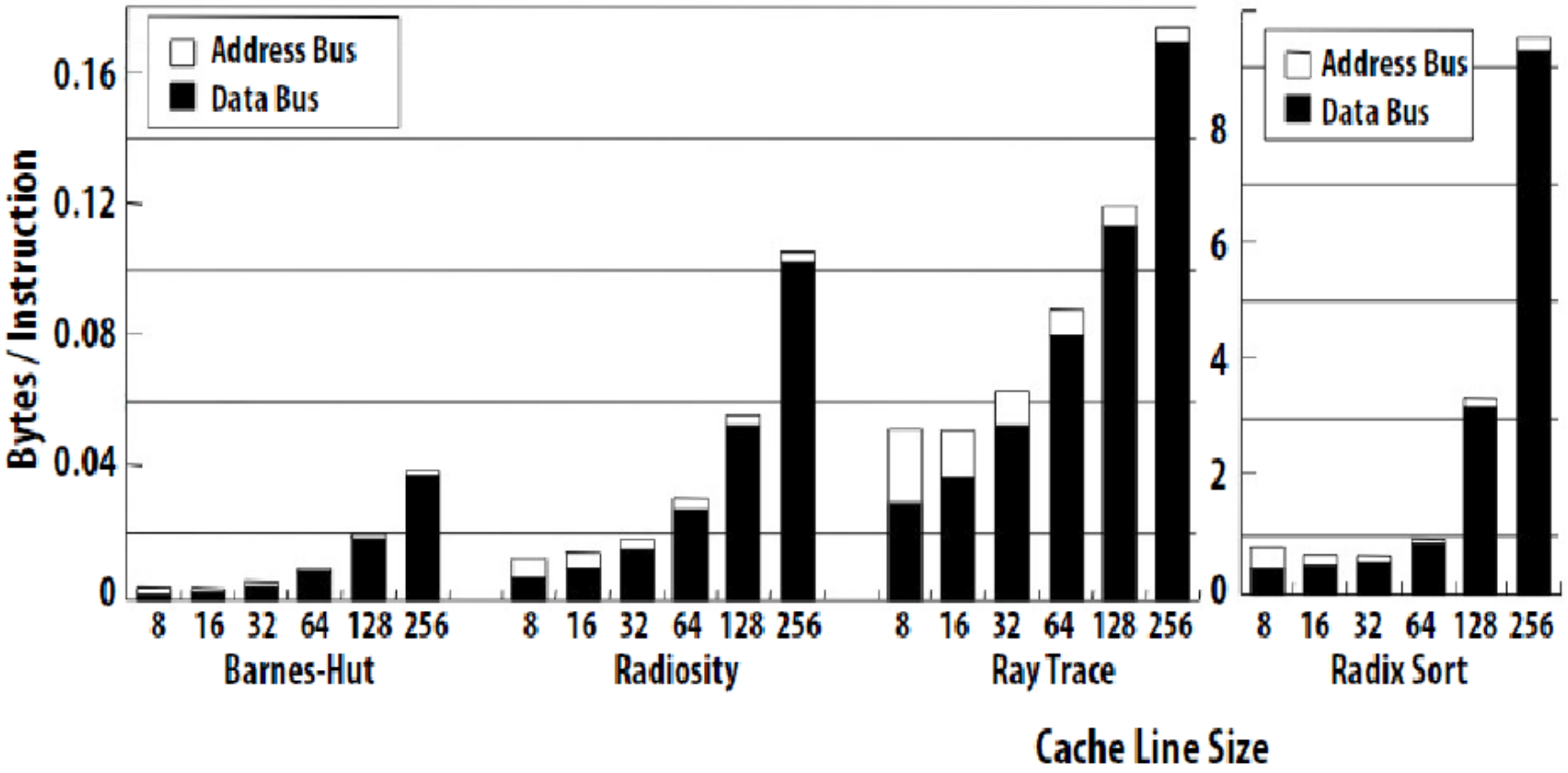
Uticaj veličine bloka



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

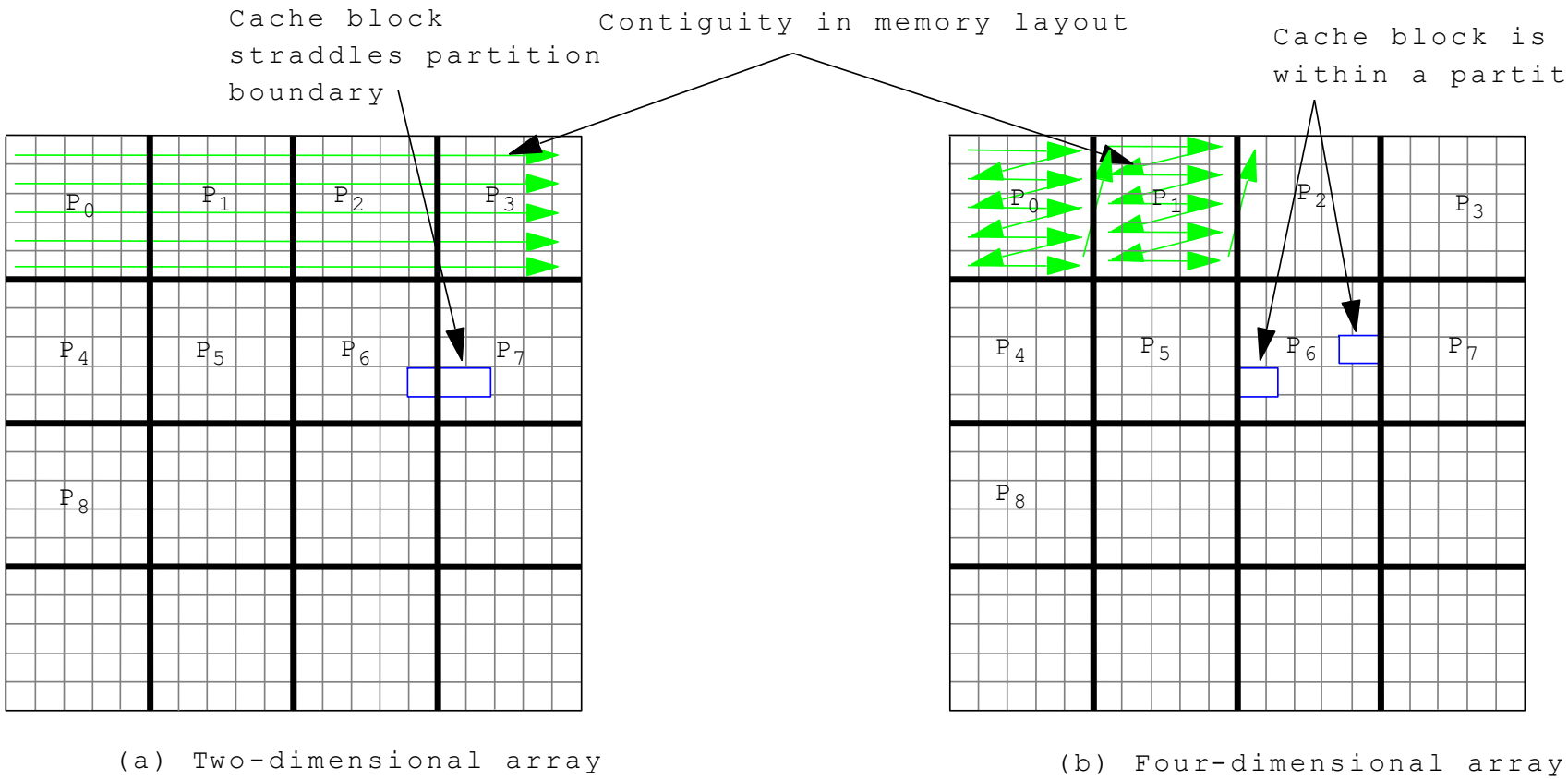
Implikacije za SW

- Izbegavanje migracije procesa
  - O tome vodi računa OS
  - Logički privatne podatke može učiniti fizički deljenim
    - Ukoliko se proces premesti na drugi procesor
    - Izaziva se nepotreban *overhead* održavanja koherencije
- Izbegavanje deljenja sa upisom (*write sharing*) ili ga, bar, vremenski lokalizovati
  - Suština paralelnih aplikacija
  - Vremenski lokalizovati kroz koncept migratorne deljivosti
  - Često zahteva sinhronizaciju (još skuplje!)
  - Sinhronizacija se takođe izvodi kroz deljene promenljive

Implikacije za SW

- Smanjiti prostorno preplitanje pristupa
  - Pri raspoređivanju posla procesorima (npr. obrada niza)
  - Pri strukturiranju podataka
  - Blokovska vs. ciklična dekompozicija kod nizova i matrica
  - 4D umesto 2D podela podataka kod matrica da bi se obezbedila kontinualnost particije
- Izbegavati konfliktne promašaje
  - Iako je logička veličina strukture 2^i, alocirati veću fizičku veličinu
  - Izbegavati „računarske brojeve“
  - Može interferirati sa veličinom blokova, setova, keša...
- Kopirati razdvojene podatke pre korišćenja
  - Moguća interferencija sa podacima drugih procesora
  - Privremene kontinualne strukture (cena kopiranja!)

Implikacije za SW



- Problem ukoliko fizički blok prevazilazi logičke nadležnosti procesora
  - Nepotrebne invalidacije na ivicama bloka

Implikacije za SW

- Koristiti odvojene "*heap*"-ove za procese
  - Da se izbegne mešanje dinamičkih podataka
  - Izbegavanje efekta lažnog deljenje
- Organizacija niza zapisa
  - Primer aplikacije koja obrađuje čestice
  - Zapis - pozicija, brzina i sile po sve tri koordinate (*x,y,z*)
  - Tri procesa nezavisno obrađuju poziciju, brzinu i sile
  - Procesori koriste različite podatke
  - Podaci mogu biti u više blokova
  - Nastaju *overhead-i* zbog održavanja koherencije usled lažnog deljenja
  - Rešenje – reorganizacija niza zapisa u tri odvojena niza
  - Posebni nizovi za poziciju, brzinu i sile

Implikacije za SW

- Vezati niz za početak bloka (*alignment*)
  - Izbegavati da element niza pripada različitim blokovima
  - Npr. problem ako su elementi veličine bloka
- Dopunjavati elemente niza na veličinu bloka (*padding*)
  - Sprečiti da se u jednom bloku obrađuju elementi različitih procesa

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
  - Nema definitivnog pobednika
  - Obe strategije imaju svoje prednosti i nedostatke
- Osnovni kriterijum – procesorska lokalnost
  - Karakteristična za multiprocesorske sisteme
  - Indikator - dužina upisnog niza (*write run*)
  - Upisni niz je niz uzastopnih upisa jednom podatku od strane jednog procesora koji nije prekinut od strane nekog drugog procesora
    - Bilo da se radi o čitanju ili upisu u isti podatak
  - Npr. procesori imaju sledeću sekvencu pristupa:
    - P0 W x, P0 R x, P0 R x, P0 W x, P0 R x, P0 W x, P1 R x
  - Upisni niz od tri upisa za P0

Poništavanje ili ažuriranje

- Poništavanje - dobro za duže upisne nizove
  - Jednom poništi kopije – prvi upis obezbedi privatnost
  - Lokalni, jeftini upisi u stanju M bez izlaska na magistralu
  - Loše za kraće upisne nizove i deljenje tipa 1 *proizvođač – "mnogo korisnika"*
    - Česti invalidacioni promašaji (može i "*read snarfing*")
  - "Čisti" keš memoriju od zaostalih kopija koje se ne koriste
  - Cena: invalidacioni promašaji
- Ažuriranje - bolje za kraće upisne nizove
  - Cena operacije, obično, ista kao i za invalidaciju
    - Adresa, komandni signal + promenjena reč
  - Lošije za duže upisne nizove i migraciju procesa
    - Potrebna samo poslednja ažurirana vrednost

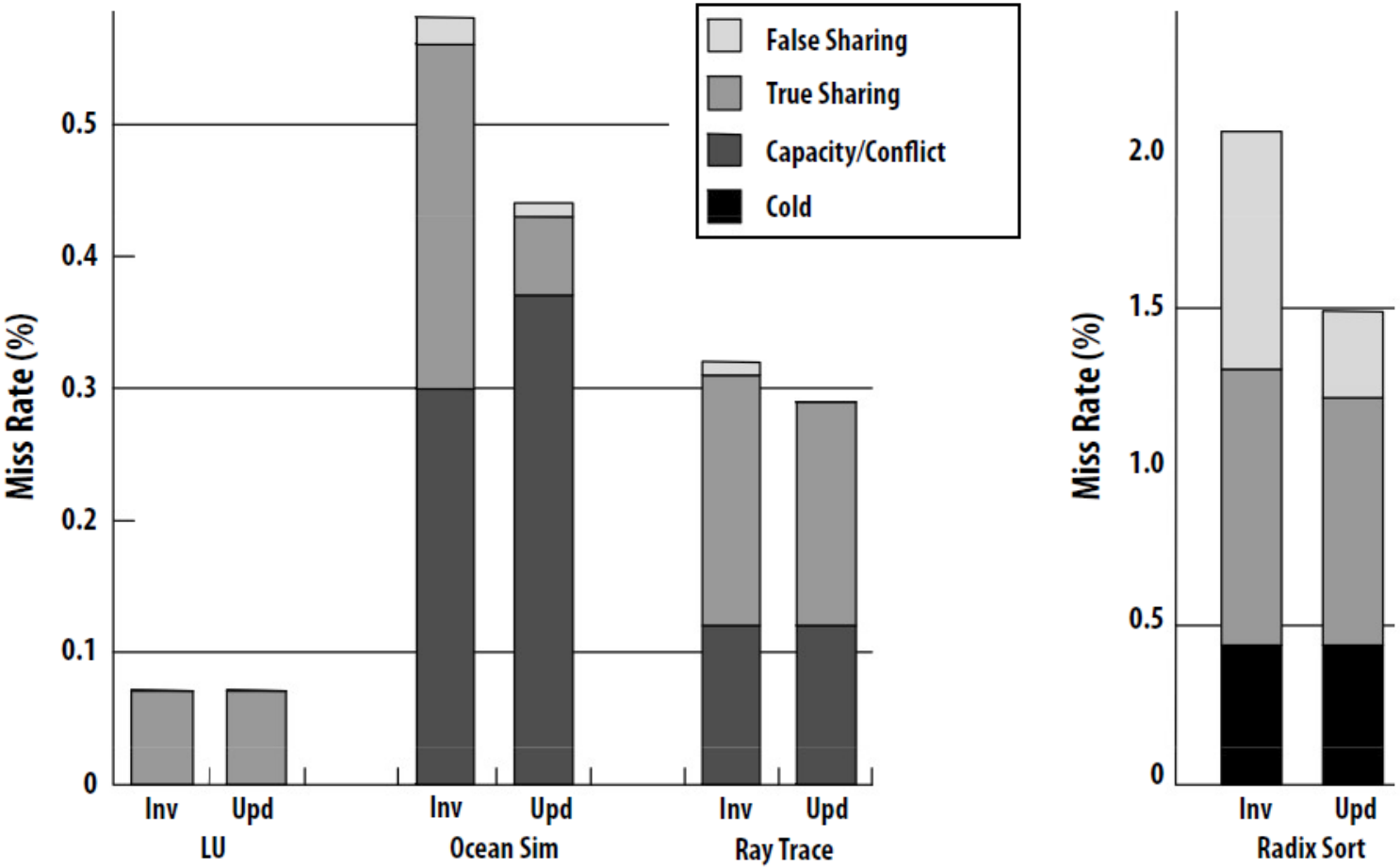
Izbor strategije upisa - primer

- Dva scenarija – šablona deljenja
  - Procesori rade nad jednom zajedničkom promeniljivom
  - Keševi na početku prazni
- SP1: {P1: Write V; P2..n: Read V} x k
  - n procesora i n keševa
  - Deljenje „1 na n“, kratak upisni niz
- SP2: {(P1: Write V) x m; P2: Read V} x k
  - Dva procesora i dva keša
  - Deljenje „1 na 1“, dugačak upisni niz
- Pretpostavke o sistemu:
  - n = 16 (broj procesora), m = 10 (broj upisa), k = 10 (broj ponavljanja scenarija)
  - inv/upg = 6 bajta (5 adresnoj + 1 cmd magistrali)
  - upd = 14 bajta (5 adr + 1 cmd + 8 word)
  - miss = 70 bajta (5 adr + 1 cmd + 8x8 = 64 block)

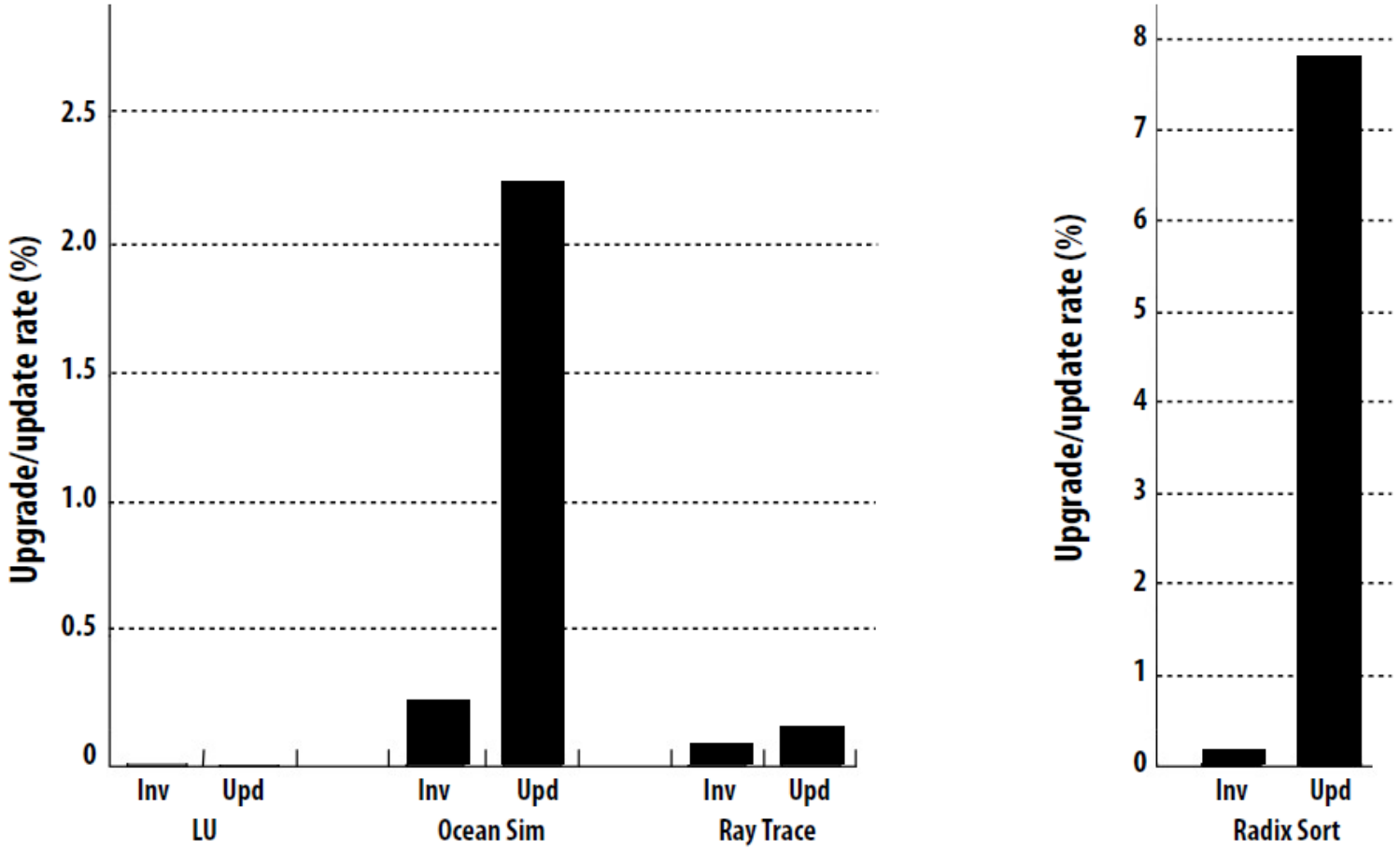
Izbor strategije upisa - primer

- SP1: {P1: Write V; P2..n: Read V} x k
- SP2: {(P1: Write V) x m; P2: Read V} x k
- WU – strategija ažuriranja
  - SP1: n x RdMiss + (k-1)x upd = 1260 bajta
    - RdMiss za dovlačenje bloka, posle samo ažuriranja svih kopija
  - SP2: 2 x RdMiss + m x (k-1)x upd = 1400 bajta
    - Ažuriranje nakon svakog upisa
- WI – strategija invalidiranja
  - SP1: (n + (k-1)x(n-1)) x RdMiss + (k-1)x upg = 10624 bajta
    - RdMiss za dovlačenje bloka, invalidacija u svih k-1 iteracija nakon upisa i RdMiss u svih k-1 iteracija za ostale procesore
  - SP2: (2 + k-1)x RdMiss + (k-1)x upg = 824 bajta
    - 2 x RdMiss za dovlačenje bloka, invalidacija i RdMiss u svih k-1 iteracija nakon upisa

Poništavanje ili ažuriranje



Poništavanje ili ažuriranje



Savremeni Intel i AMD procesori koriste invalidaciju!

Adaptivni protokoli

- Prilagođavanje ponašanju aplikacije
- 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 (ažuriranje), a drugi WI (poništavanje)
  - Novo stanje (prvi upis) i transakcija
  - Prerana invalidacija (za potrebe sinhronizacije)
    - Primer sinhronizaciona promenljiva
- EDWP (*Efficient Distributed Write Protocol*)
  - Archibald - UoW
  - Invalidacioni prag – 3 upisa (na osnovu rezultata simulacije)
  - Tri dodatna stanja – *Sco*, *Rw1* i *Rw2*
    - *Shared-clean-owned* i *Remote write* 1/2
  - 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