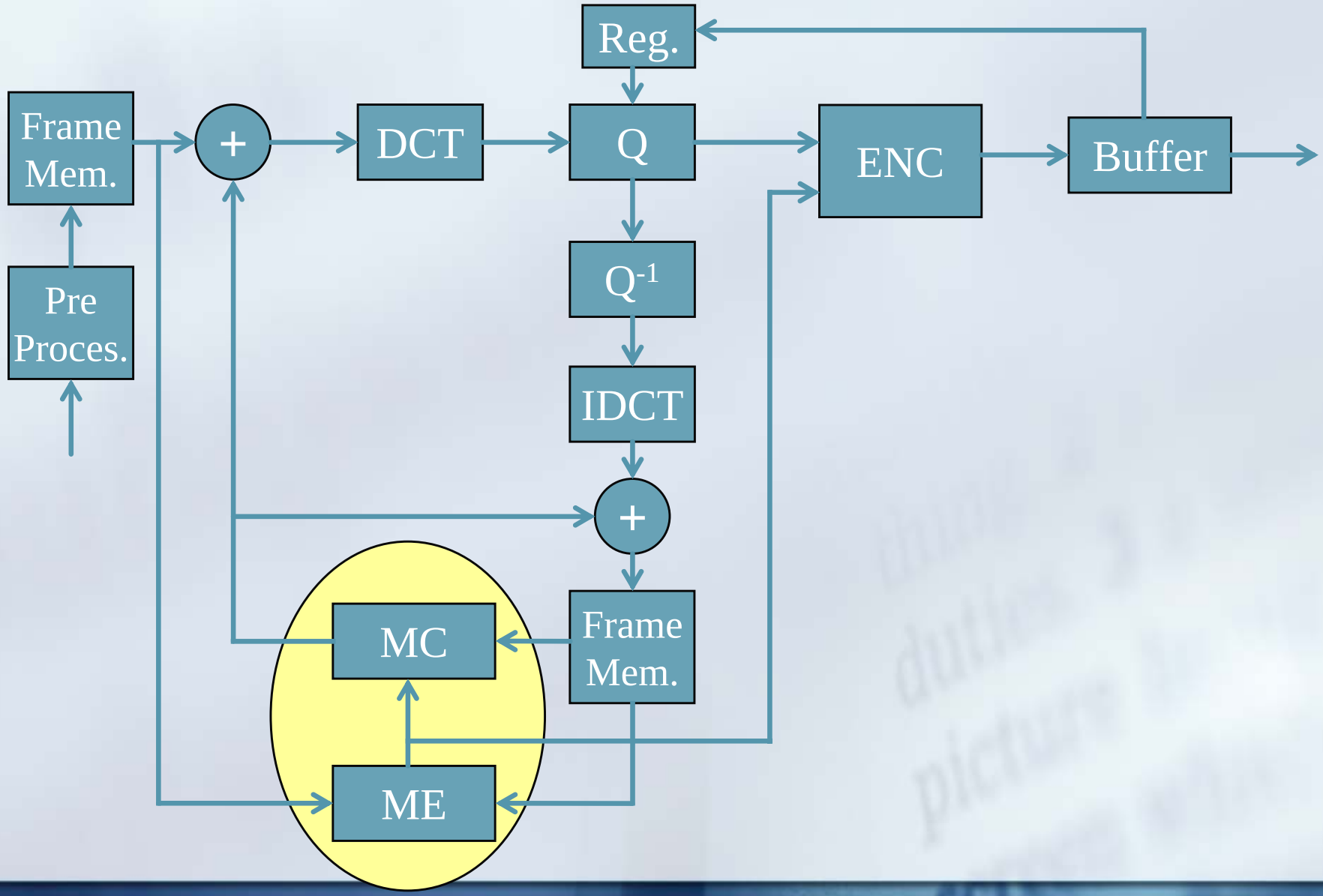
Multimedijske tehnologije

Prof.dr.sc. Mario Kovač

MPEG koder


Reg.
Frame Mem.
+
DCT
Q
ENC
Buffer
Pre Proces.
Q-1
IDCT
+
MC
Frame Mem.
ME

Intra i inter-blokovska kompresija

- **Unutar-blokovska** kompresija
  - Uklanjanje informacija visokih frekvencija koje ljudsko oko ne može prepoznati
  - Isto kao kod kompresije slika (JPEG) što je prije objašnjeno
- **Među-blokovska** kompresija
  - Smanjivanje vremenske redundancije

- Obje se metode zasnivaju na karakteristikama ljudskog vizualnog sustava (HVS)

Međublokovska kompresija

- Često se pojedini dijelovi slike unutar niza vrlo malo mijenjaju ili su čak nepromijenjeni
- Pozadina slike često se ne mijenja

prednji plan (dinamičan) pozadina (statična)

- Između susjednih slika sekvence nema mnogo promjene ✸ dijelovi slike se ponavljaju ✸ **vremenska redundancija**
- Dijelovi koji se mijenjaju mogu se analizirati kao **pokreti objekata** ✸ pokreti opisuju promjene

Vremenska redundancija

Vremenska redundancija

Vremenska redundancija

Procjena i kompenzacija pokreta

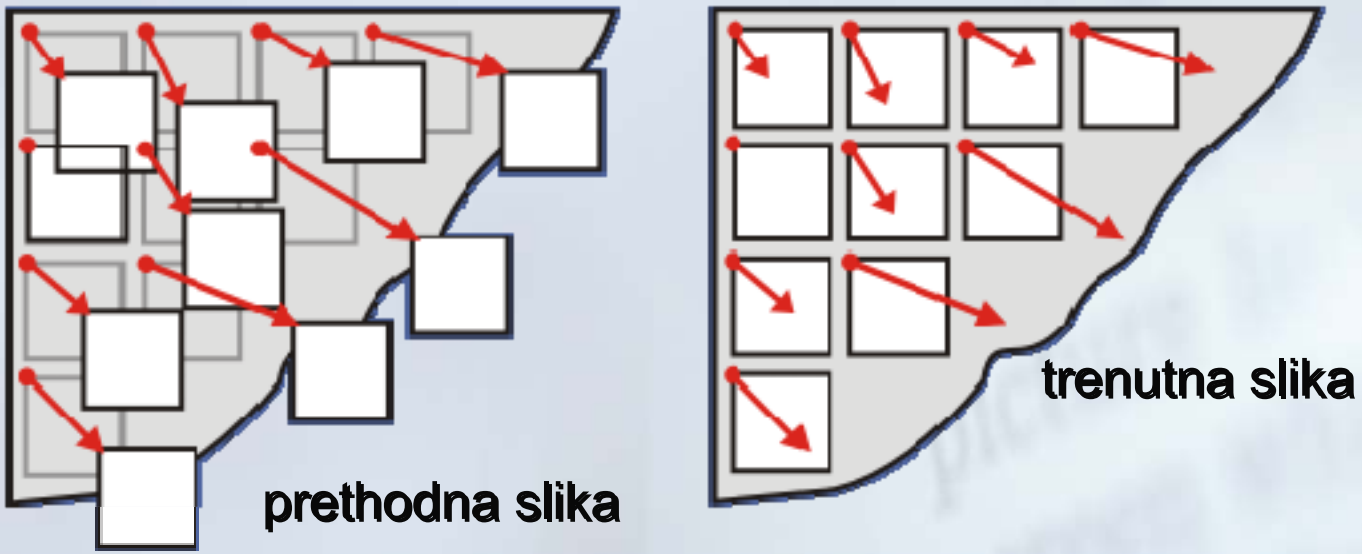
- **Procjena pokreta (motion estimation, ME):**
  - Izdvajanje dijelova slike (segmentacija pokreta)
  - Opisivanje pokreta (procjena)
  - Izvodi se u koderu
- **Kompenzacija pokreta (motion compensation, MC):**
  - Korištenje rezultata procjene pokreta
  - Izvodi se u dekoderu

Procjena i kompenzacija pokreta

- Kako bi omogućili jednostavniju implementaciju nameću se potrebna pojednostavljenja:
  - Segmentacija na pravokutne dijelove unaprijed znanih dimenzija (blok)
  - Svaki je pokret translacijski (vektor pomaka)

Procjena i kompenzacija pokreta

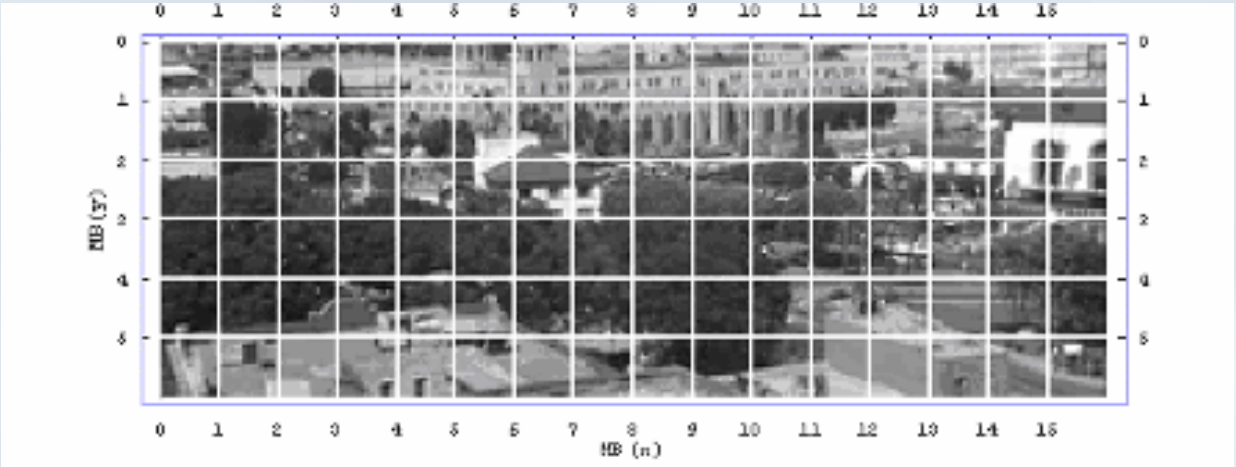
- Algoritmi procjene pokreta za svaki blok računaju pripadni **vektor pomaka**
- Vektor pokazuje izvorište bloka u prethodnoj slici prije pomaka na poziciju u trenutnoj slici



Procjena i kompenzacija pokreta

Slika kao pojedinačni objekt pri kompresiji

- Radi efikasnije obrade za procjenu pokreta dovoljna je **komponenta Y**



Mjera poremećaja

- Mjera sličnosti dvaju blokova jest **mjera poremećaja** slike od jednog trenutka do drugog
- Procjena pokreta svodi se na **optimizaciju** mjere poremećaja kao kriterijske funkcije
- Algoritmi u definiranom **području pretraživanja** nastoje pronaći minimum kriterijske funkcije
  - Promjena slike na tom je mjestu najmanja
  - proglašavamo da se blok pomakao duž vektora

Mjera poremećaja – primjer

- Neke moguće mjere poremećaja dvaju blokova:
  - Srednja kvadratna pogreška (MSE) - računski zahtjevna

$$MSE(x,y;d_x,d_y)=\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\left[Y(x+i,y+j,t_1)-Y(x+d_x+i,y+d_y+j,t_0)\right]^2$$

  - Srednji apsolutni poremećaj (MAD) – široko prihvaćena

$$MAD(x,y;d_x,d_y)=\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\left|Y(x+i,y+j,t_1)-Y(x+d_x+i,y+d_y+j,t_0)\right|$$

  - Broj podudarajućih slikovnih elemenata (MPC)

$$MPC(x,y;d_x,d_y)=\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}T\left[Y(x+i,y+j,t_1),Y(x+d_x+i,y+d_y+j,t_0)\right]$$

$$T(\alpha,\beta)=\begin{cases}1, & \text{za } |\alpha-\beta|\le T_0\\ 0, & \text{inace}\end{cases}$$

Algoritmi

- Ovisno o načinu pretraživanja razlikujemo:
  - **Algoritam potpunog pretraživanja**
    - pretražuje sve točke unutar područja pretraživanja
    - računski vrlo zahtjevan
    - Daje najbolji mogući rezultat
  - **Nepotpuni (brzi, napredni) algoritmi**

Algoritam potpunog pretraživanja

- Veoma zahtjevan algoritam za računalne resurse:
  - Izračun jedne vrijednosti poremećaja (MAD):
    - Dohvat podataka (vrlo zahtjevno)
    - Za blok 16x16: 256 oduzimanja + 1 pomak (dijeljenje sa 256)
  - Ako područje pomaka iznosi +-8 u obje dimenzije:
    - (2*8+1)*(2*8+1)*(256+dohvat)=73984 + dohvati
    - Pronalaženje minimuma
  - Za jednu sliku 1280x1024
    - 5120*73984=378.798.080 + dohvati
  - Za 30 slika u sekundi: $1{,}1*10^{10}$ zbrajanja/s

Algoritam potpunog pretraživanja

- Vidljivo je da algoritam potpunog pretraživanja daje optimalni rezultat ali je nažalost u praksi teško primjenjiv
- Koriste ga uglavnom samo HW koderi

- Upravo zato potreba za brzim algoritmima
  - Prednosti: manje potrebnih operacija
  - Nedostaci: veće razlike koje se trebaju kodirati a time je i izlazna količina podataka veća
  - Kvaliteta nekih algoritama zadovoljavajuća te su blizu potpunom pretraživanju
  - Problemi sa tipovima pokreta (objašnjeno kasnije)

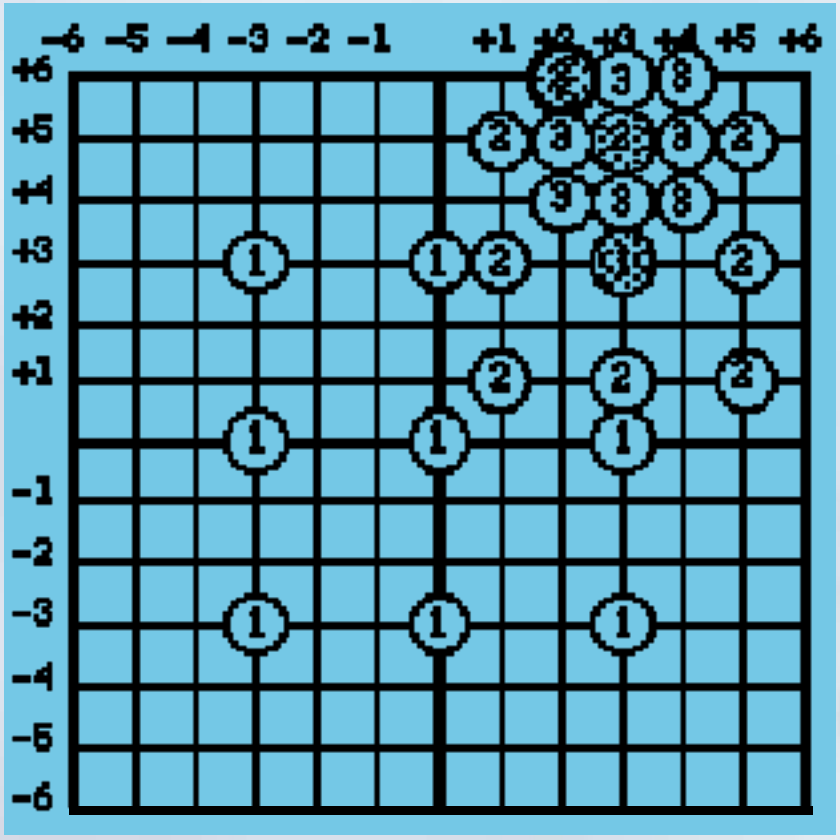
Brzi algoritmi

- **Brzi, napredni algoritmi pretraživanja** - usmjeravaju pretraživanje ovisno o vrijednosti poremećaja slike

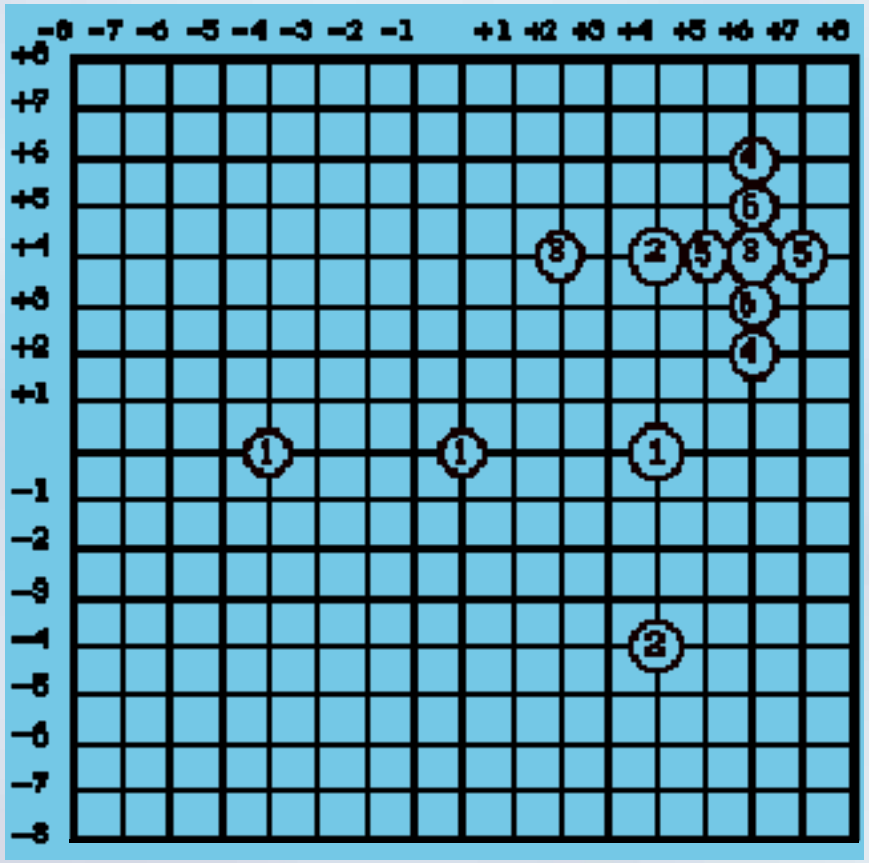
  Neki primjeri:

  - Logaritamsko pretraživanje (LOG)
  - Pretraživanje u tri koraka (3SS)
  - Ortogonalno pretraživanje (ORT)
  - Algoritam gradijentnog spusta temeljen na blokovima (BBGDS)

Algoritmi – primjer (3SS)

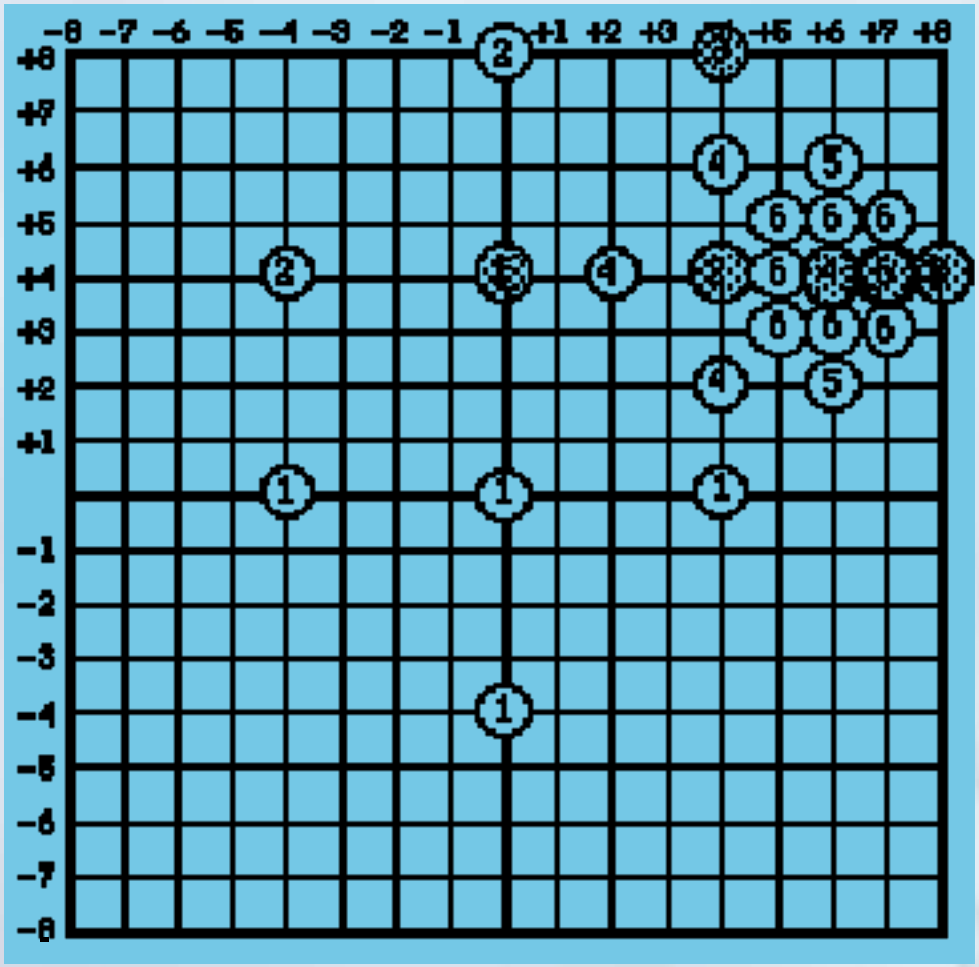


Algoritmi – primjer (ORT)

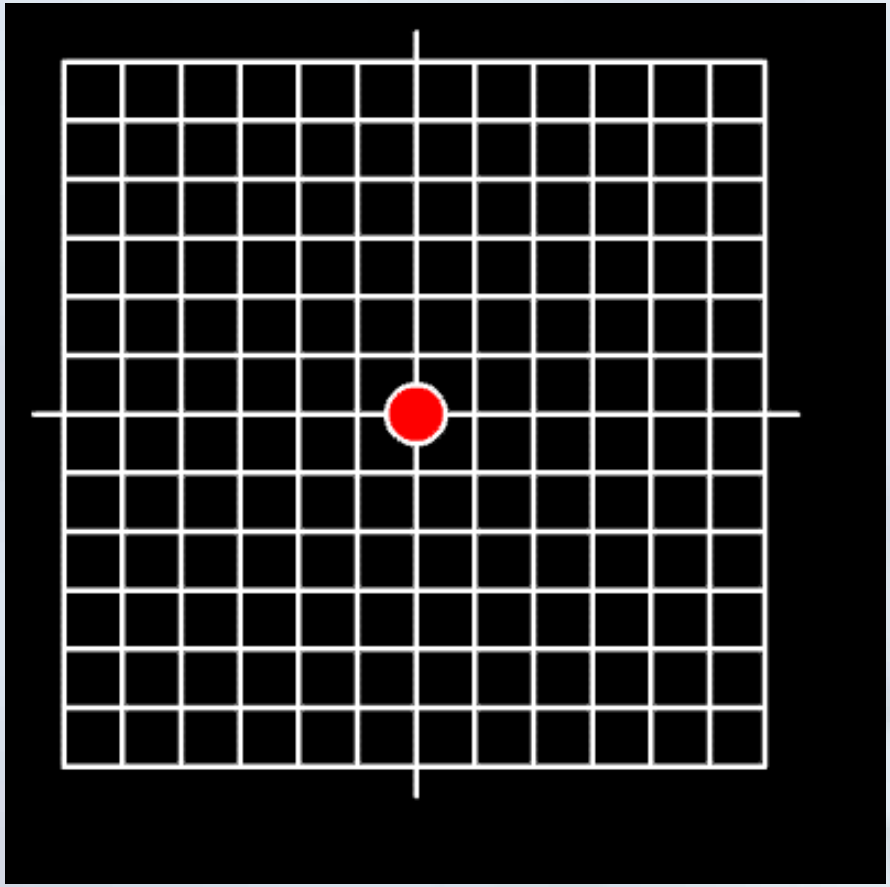


Algoritmi – primjer (LOG)

- 2D logaritamsko pretraživanje

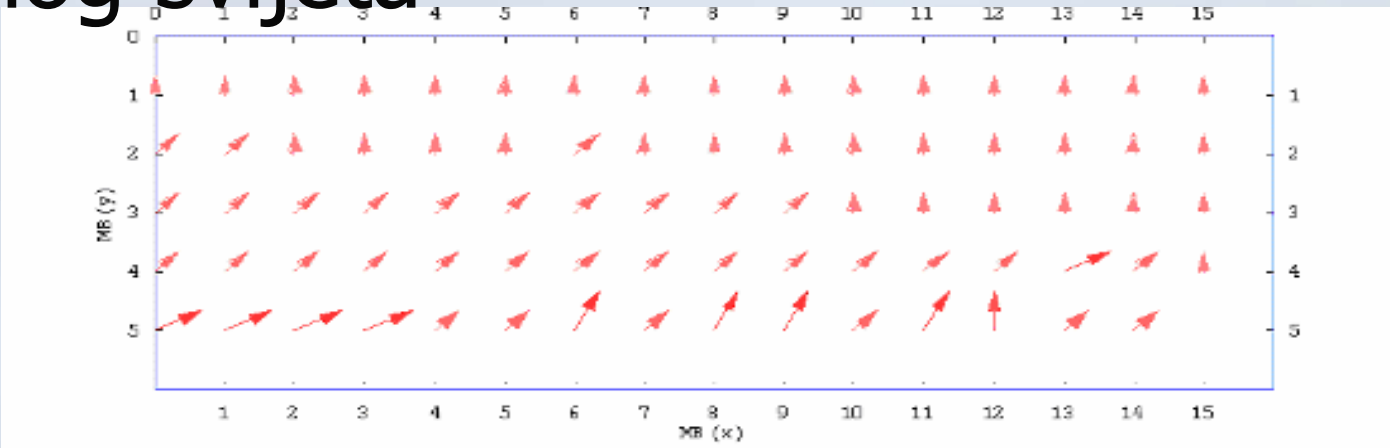


Algoritmi – primjer (LOG)



Raspodjela vektora pomaka

- Prikaz upućuje na ravnomjerno raspoređene pokrete između dvije slike sekvence
- Vektori su većinom kratki - centrirana raspodjela vektora - tipična za sekvence iz stvarnog svijeta

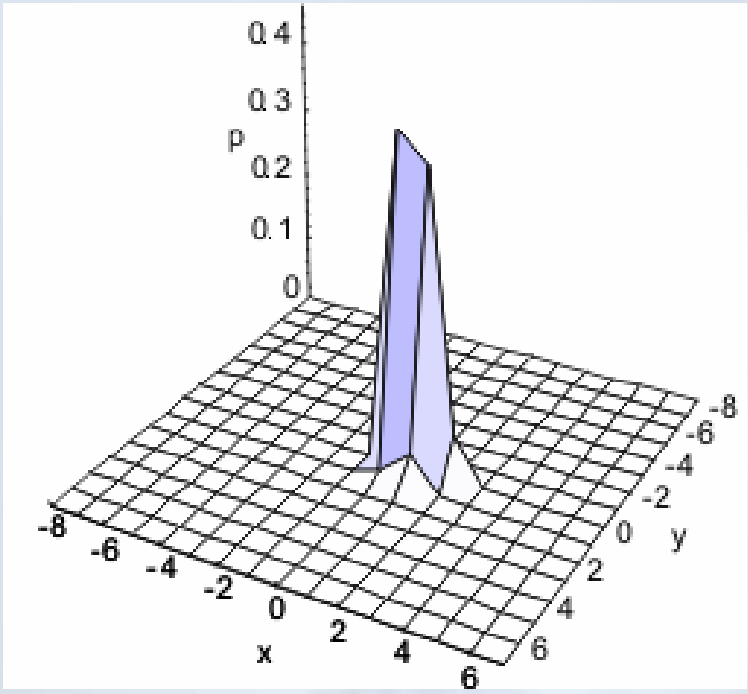


Raspodjela vektora (2)

- Prebrojavanje vektora za sve moguće parove komponenti (x,y) tijekom cijele sekvence daje relativne frekvencije vektora

- Učinkovitost algoritma moguće je povećati ako se u obzir uzme uočena centrirana raspodjela vjerojatnosti vektora

- središtu područja pretraživanja treba posvetiti više pažnje



Primjeri usporedbe algoritama

- Parametri koji određuju učinkovitost algoritma:
  - Ukupna **mjera poremećaja** nakon procjene pokreta treba biti što manja (bolja kompresija)
  - Ukupni broj **ispitnih točaka** također treba biti što manji (veća brzina)
- Potreban je kompromis **kompresija** ↔ **brzina**
- Primjer tri sekvence koje sadržavaju različite vrste pokreta:
  - *"City View"* - ujednačeno raspoređen i malen pokret
  - *"Car Jump"* - lociran i velik pokret
  - *"Troops"* - ujednačeno raspoređen i velik pokret

“City view”

- Ujednačeno raspoređeni i mali pokreti

“Car jump”

- Locirani i veliki pokreti

“troops”

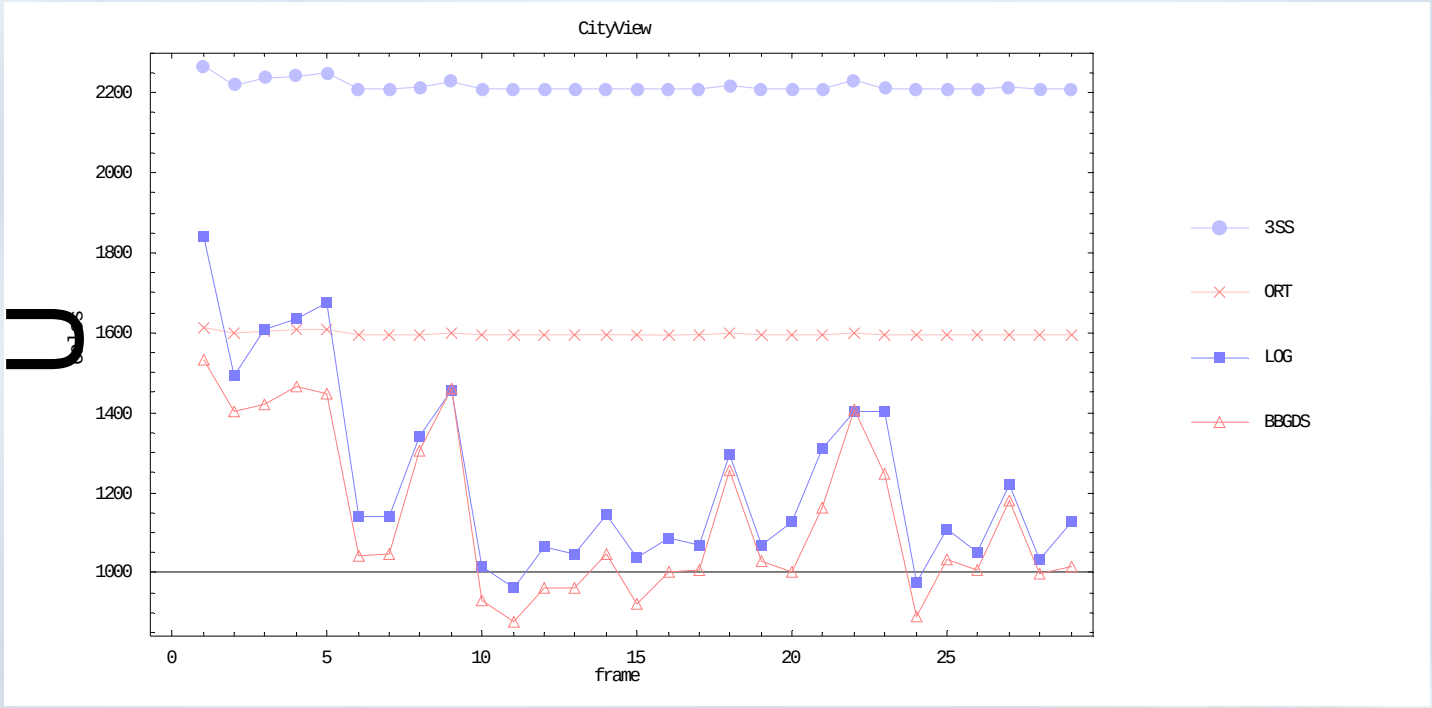
- Ujednačeno raspoređeni i veliki pokreti

Implementacija – primjer

- Norma za kompresiju **ne specificira** postupke procjene pokreta
- Kvaliteta aplikacije znatno ovisi o načinu procjene pokreta

Rezultati (1)

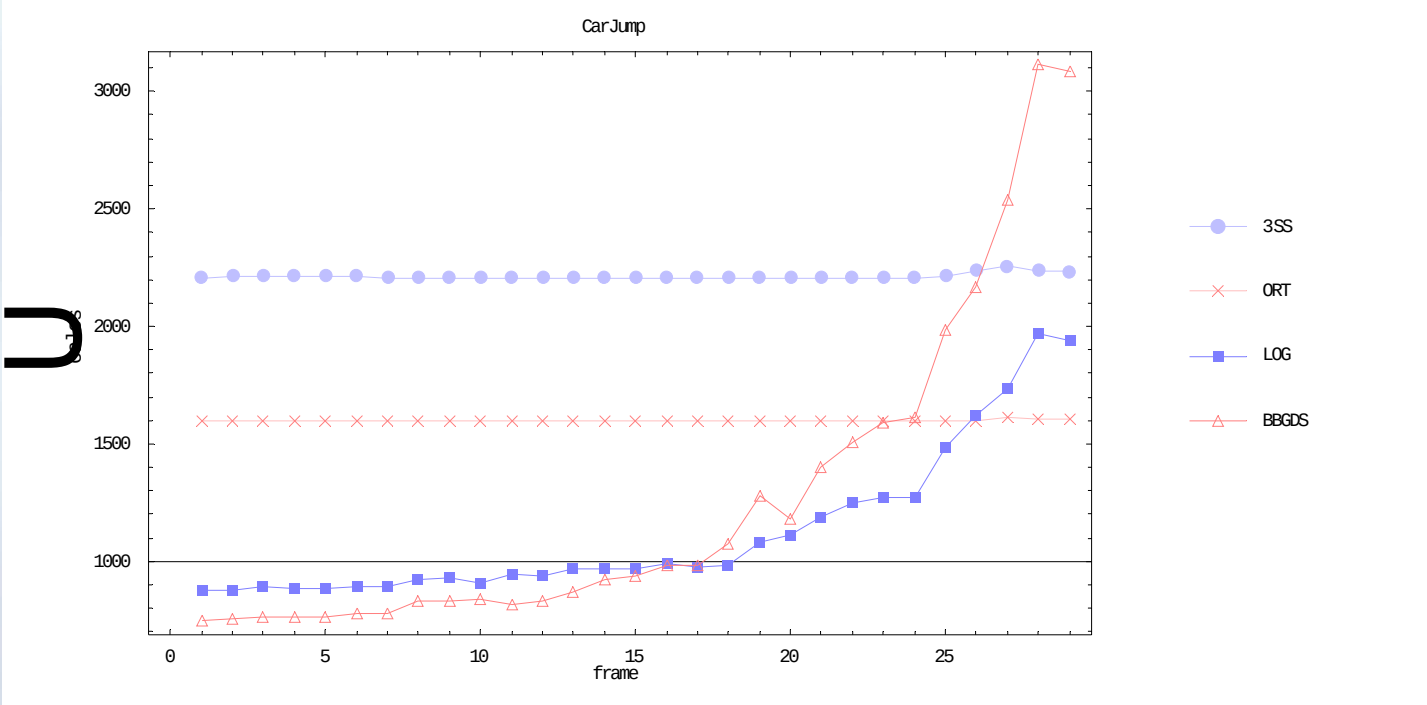
- "City View" sekvenca – broj ispitnih točaka:



- Centrirana raspodjela: BBGDS najbolji
- ORT i 3SS konstantan i relativno velik broj ispitnih točaka

Rezultati (2)

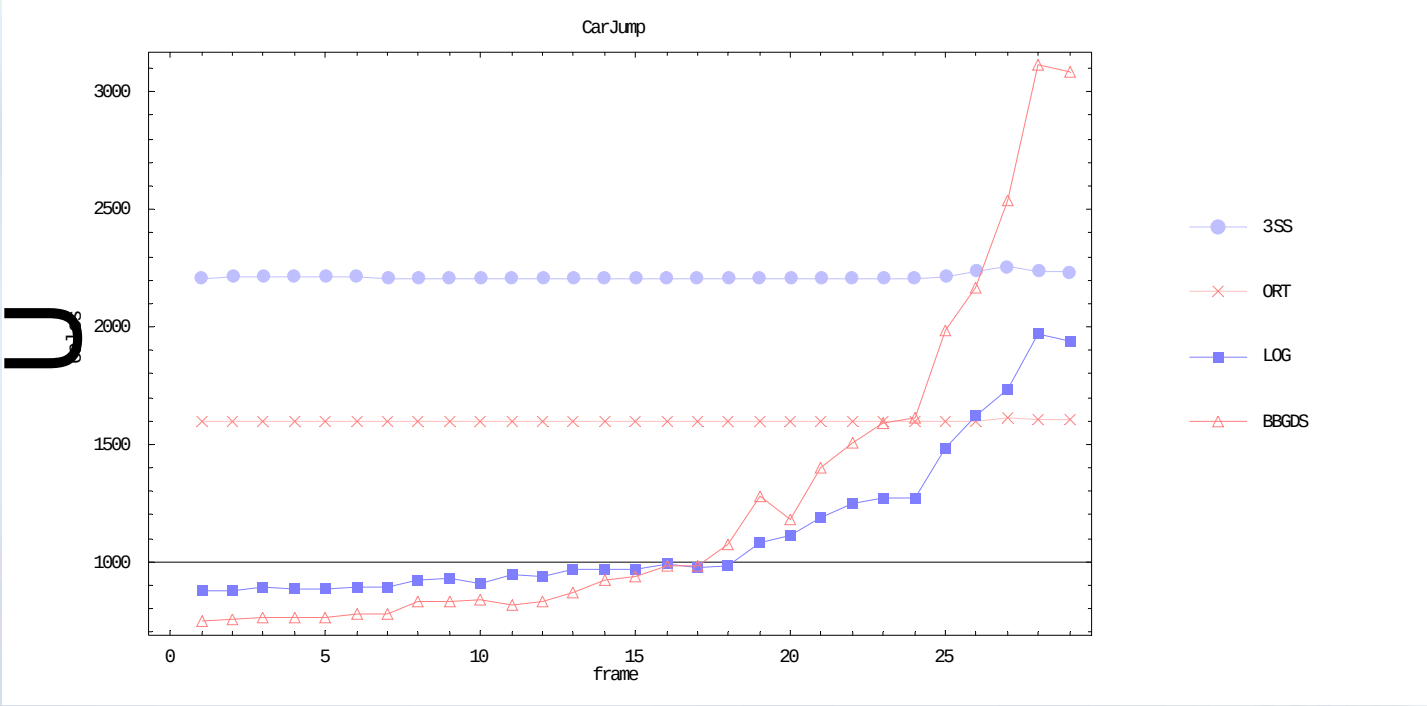
- "Car Jump" sekvenca – broj ispitnih točaka:



- Lokalizirana rastuća promjena: BBGDS i LOG daju slabije rezultate kako poremećaj raste
- ORT bolji od 3SS, konstantni unatoč poremećaju

Rezultati (3)

- "Troops" sekvenca – broj ispitnih točaka:



- Brojni složeni pokreti: BBGDS loš radi veće prosječne duljine vektora pomaka
- LOG vrlo dobar prema oba kriterija

Rezultati (4)

- Učinkovitost pojedinog algoritma ovisi o vrsti sadržanog pokreta
- Učinkovit postupak će na temelju vrste pokreta heuristički odabrati najprikladniji algoritam: **adaptivni algoritmi**

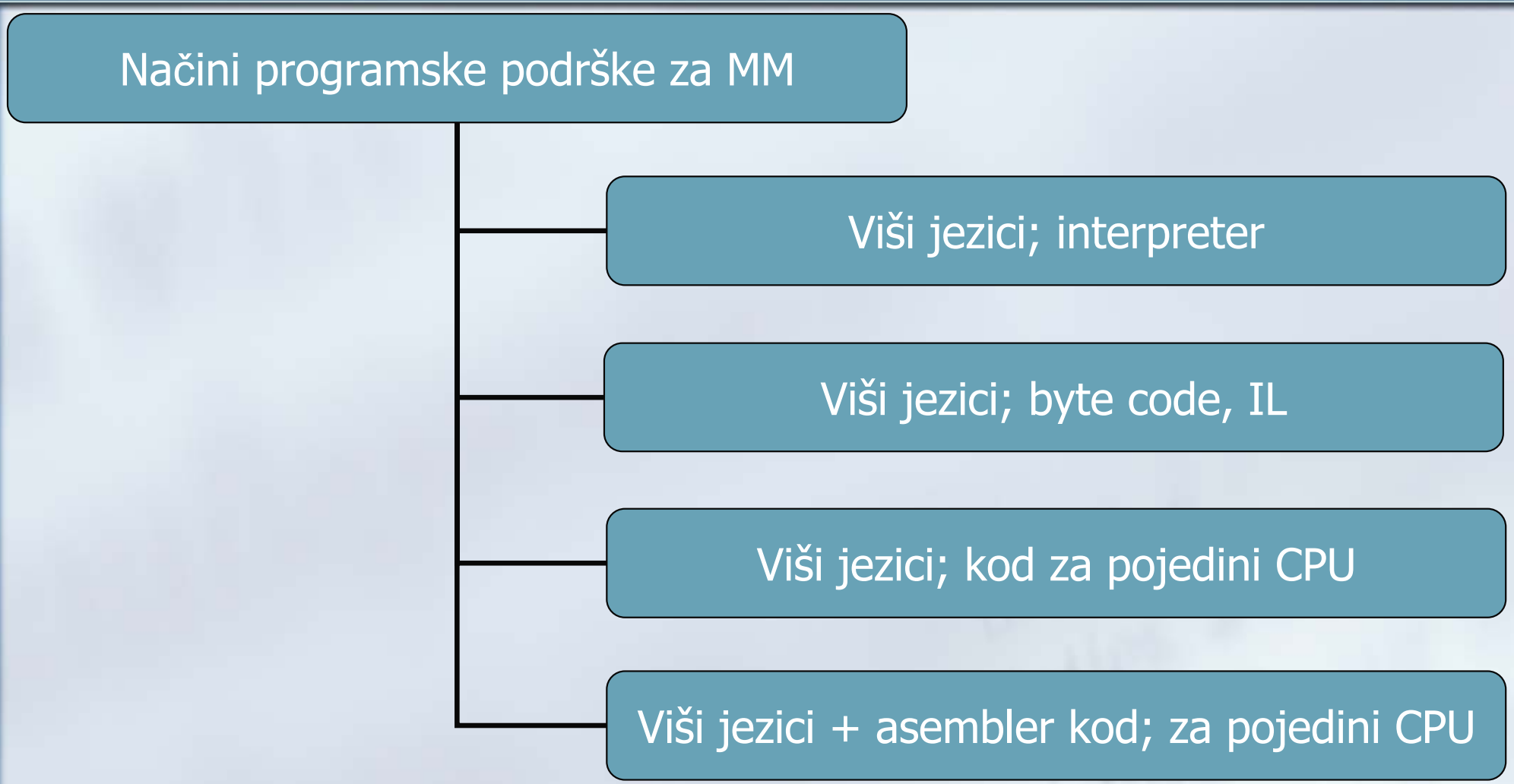
Izvedba

- Ovime smo ukratko analizirali neke najzahtjevnije algoritme pri obradi i kompresiji multimedijskih podataka
- Vidjeli smo i načelnu kompleksnost izračuna bez ulaženja u previše detalja
- Postavlja se pitanje kako efikasno izvesti takve algoritme

Optimizacije

- Podsjetimo se sa prošlog predavanja na neke osnovne grupe optimizacija:
  - Smanjenje preciznosti izračuna
  - Razvoj ekvivalentnih matematičkih algoritama sa manjom kompleksnosti
  - Promjena programske razvojne okoline
  - Promjena programske izvedbene okoline
  - Promjena arhitekture sustava za izvođenje

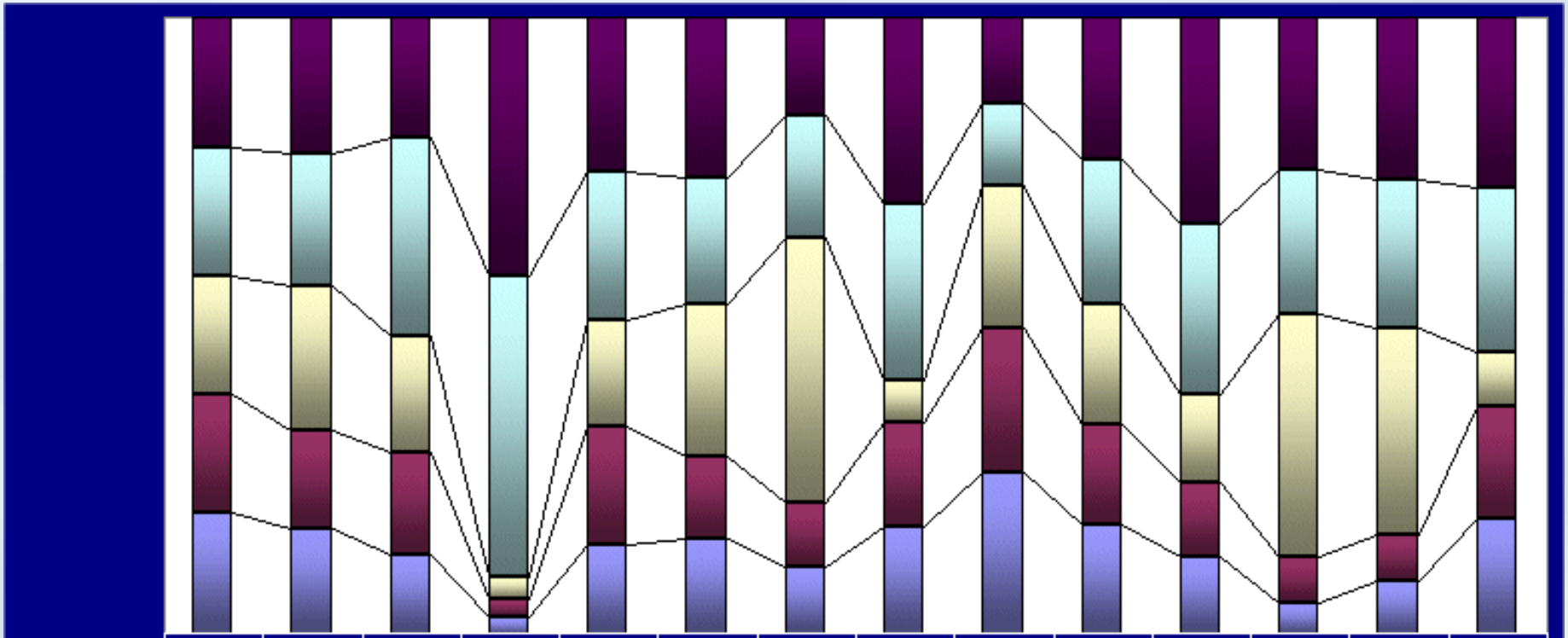
Osnovni načini SW podrške za MM



Performanse

- Morate biti svjesni da se program pisan u Javi ili npr. C# sporije izvodi od programa koji su prevedeni u kod za ciljni procesor
- Dodatno: viši programski jezici generiraju sporiji kod od onoga pisanog u kombinaciji sa asemblerom

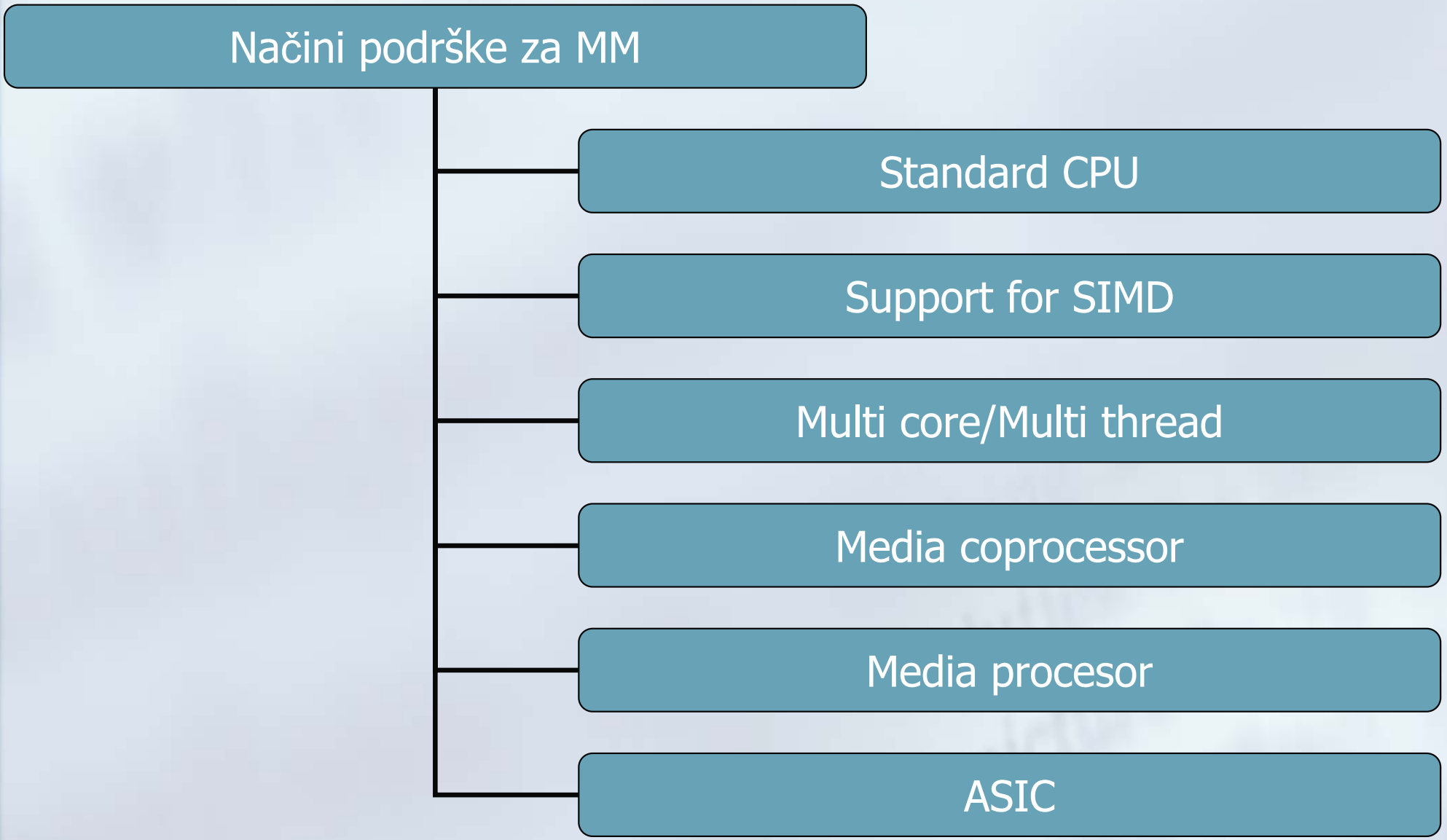
Usporedba Java, C#, C++



	int(32 bit) math	long(64 bit) math	double math	trig. Math	I/O	array	exceptions	hashmap	hashmaps	heapsort	vector	matrix multiply	nested loops	string concat. (fixed)
java 1.5 alpha	7922	20422	9609	39656	9109	2281	3985	1047	4562	1485	32593	28954	19297	1781
java 1.4.2_03	7938	20079	15718	46406	8859	1781	4844	1000	4375	1516	27031	27453	17453	1719
c#	7250	21500	9296	3187	6234	2156	10703	234	7593	1250	14015	46421	24586	578
cpp (MS)	7171	15031	8110	2922	7062	1157	2547	593	7656	1062	11953	8750	5438	1172
cpp (Intel)	7406	15531	6281	2438	5187	1328	2668	594	8562	1125	11938	5781	6094	1203

www.tommti-systems.de/main-Dateien/reviews/languages/benchmarks.html

Osnovni načini CPU podrške za MM



Standardni CPU

- Ako se prisjetimo Arhitekture računala onda znamo da obični procesori mogu izvesti jednu ALU operaciju po periodu
- Operacija je širine koju ima ALU

- Kako bi ubrzali izvođenje moramo mnogo pažnje posvetiti dohvatu podataka te optimizacijama petlji

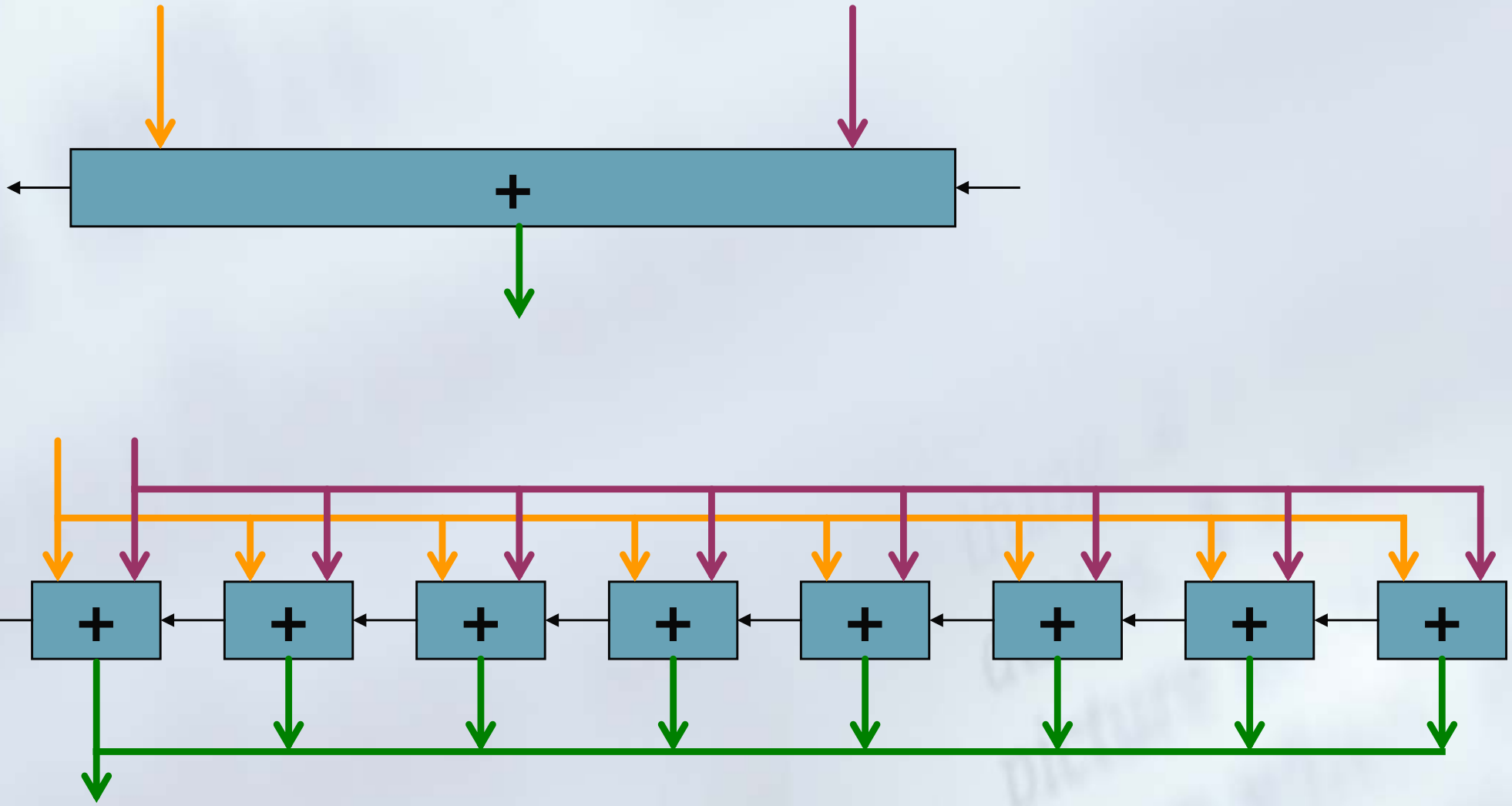
Standardni CPU

- Za DSP operacije (npr DCT) izuzetno je korisno ako procesor ima sklopovski izvedeno množenje i pripadnu naredbu
- Dodatna značajna prednost ako postoji naredba množenja sa zbrajanjem (Multiply Accumulate)
  - Kod primjera računanja DCT, DFT i slično imamo većinu "leptir" operacija kod kojih je MLA osnovna karika

Podrška za SIMD

- SIMD (Single Instruction Multiple Data)
  - Arhitektura puta podataka u procesoru koja omogućuje obradu više podataka (u načelu manje preciznosti) sa jednom naredbom
- Osnovna ideja:
  - Ako imamo npr 64 bitovnu ALU onda je potpuno neefikasno s njom obrađivati 8 bitovne podatke
  - Reorganizirati ALU na način da se može "podijeliti"

ALU – obična i SIMD



MM proširenja

- Ovo (osnovno) načelo služi kako bi se obrada multimedijskih algoritama značajno ubrzala
- Primjeri:
  - Stari: VIS, PA-RISC
  - Ne tako stari: MMX, 3DNow!
  - Novi: SSE4

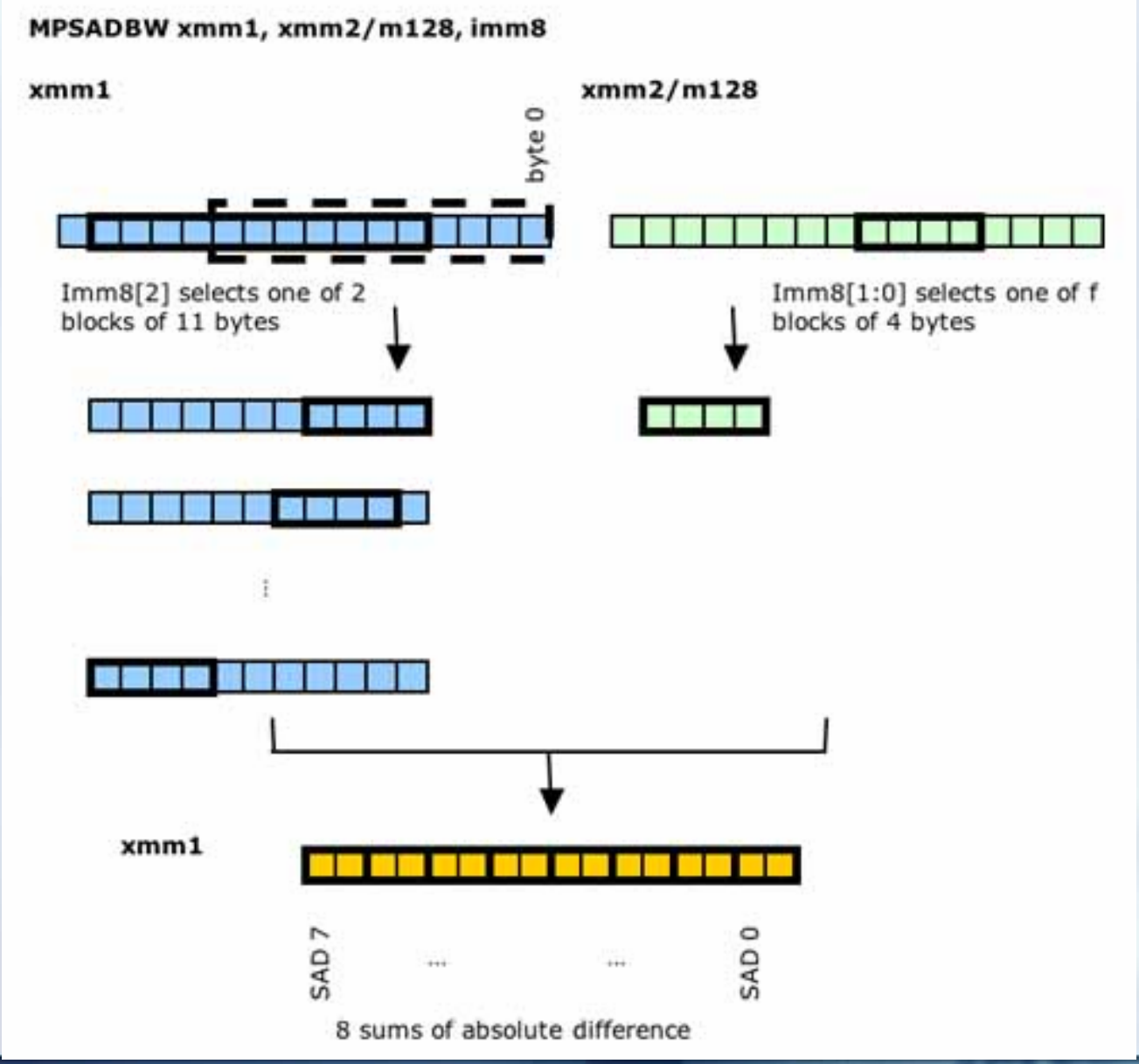
SSE4 dio skupa naredaba...

Instruction	Description
PEXTRB r32/m8, xmm, imm8	Extract Byte
PEXTRD r/m32, xmm, imm8	Extract Dword
PEXTRQ r/m64, xmm, imm8	Extract Qword
PEXTRW r/m16, xmm, imm8	Extract Word
PHMINPOSUW xmm1, xmm2/m128	Packed Horizontal Word Minimum
PINSRB xmm1, r32/m8, imm8	Insert Byte
PINSRD xmm1, r/m32, imm8	Insert Dword
PINSRQ xmm1, r/m64, imm8	Insert Qword
PMAXSB xmm1, xmm2/m128	Maximum of Packed Signed Byte Integers
PMAXSD xmm1, xmm2/m128	Maximum of Packed Signed Dword Integers
PMAXUD xmm1, xmm2/m128	Maximum of Packed Unsigned Dword Integers
PMAXUW xmm1, xmm2/m128	Maximum of Packed Unsigned Word Integers
PMINSB xmm1, xmm2/m128	Minimum of Packed Signed Byte Integers
PMINSD xmm1, xmm2/m128	Minimum of Packed Signed Dword Integers
PMINUD xmm1, xmm2/m128	Minimum of Packed Unsigned Dword Integers
PMINUW xmm1, xmm2/m128	Minimum of Packed Unsigned Word Integers
PMOVSXBD xmm1, xmm2/m32	Packed Move with Sign Extend - Byte to Dword
PMOVSXBQ xmm1, xmm2/m16	Packed Move with Sign Extend - Byte to Qword

Primjer: MPSADBW

- MPSADBW sums the absolute difference of 4 unsigned bytes, selected by bits [0:1] of the immediate byte (third operand), from the source (second operand) with sequential groups of 4 unsigned bytes in the destination operand. The first group of eight sequential groups of bytes from the destination operand (first operand) start at an offset determined by bit 2 of the immediate. The operation is repeated 8 times, each time using the same source input but selecting the next group of 4 bytes starting at the next higher byte in the destination. Each 16-bit sum is written to dest.

Primjer: MPSADBW



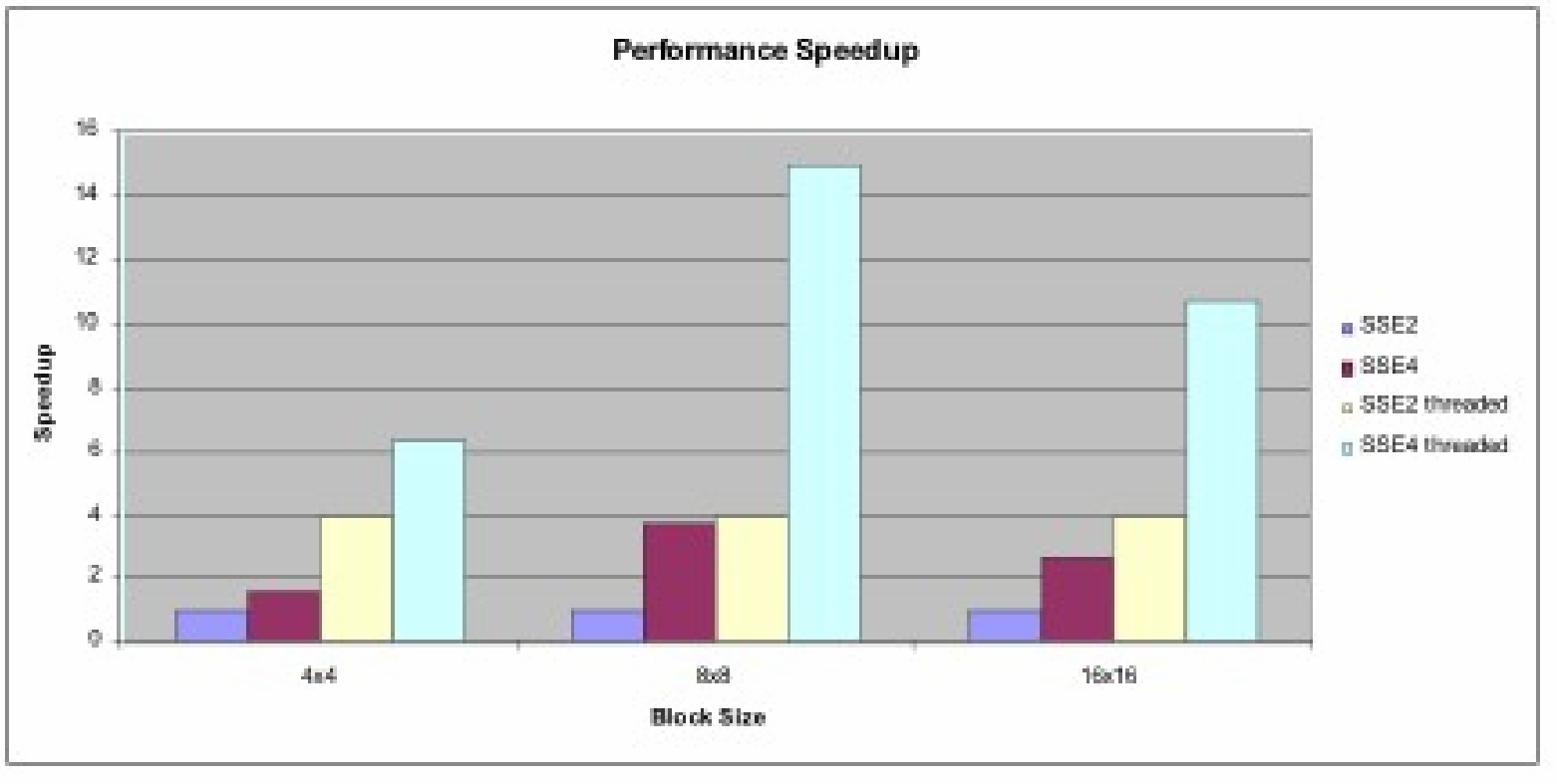
Primjer optimizacije (BlockMatch4x4)

- Neoptimiziran kod
- SSE2 optimiziran
- SSE4 optimiziran

Primjer ubrzanja

Code Sample	Cycles / Block SAD	Speedup
4x4 Block		
C++	54.84	
SSE2	4.32	1.00
Intel SSE4	2.71	1.59
8x8 Block		
C++	180.55	
SSE2	25.29	1.00
Intel SSE4	6.73	3.83
16x16 Block		
C++	173.01	
SSE2	71.42	1.00
Intel SSE4	26.86	2.66

Ubrzanje


Performance Speedup
Speedup
0
2
4
6
8
10
12
14
16
4x4
8x8
16x16
Block Size
SSE2
SSE4
SSE2 threaded
SSE4 threaded