Software Development Seminar

Graphics (Advanced)

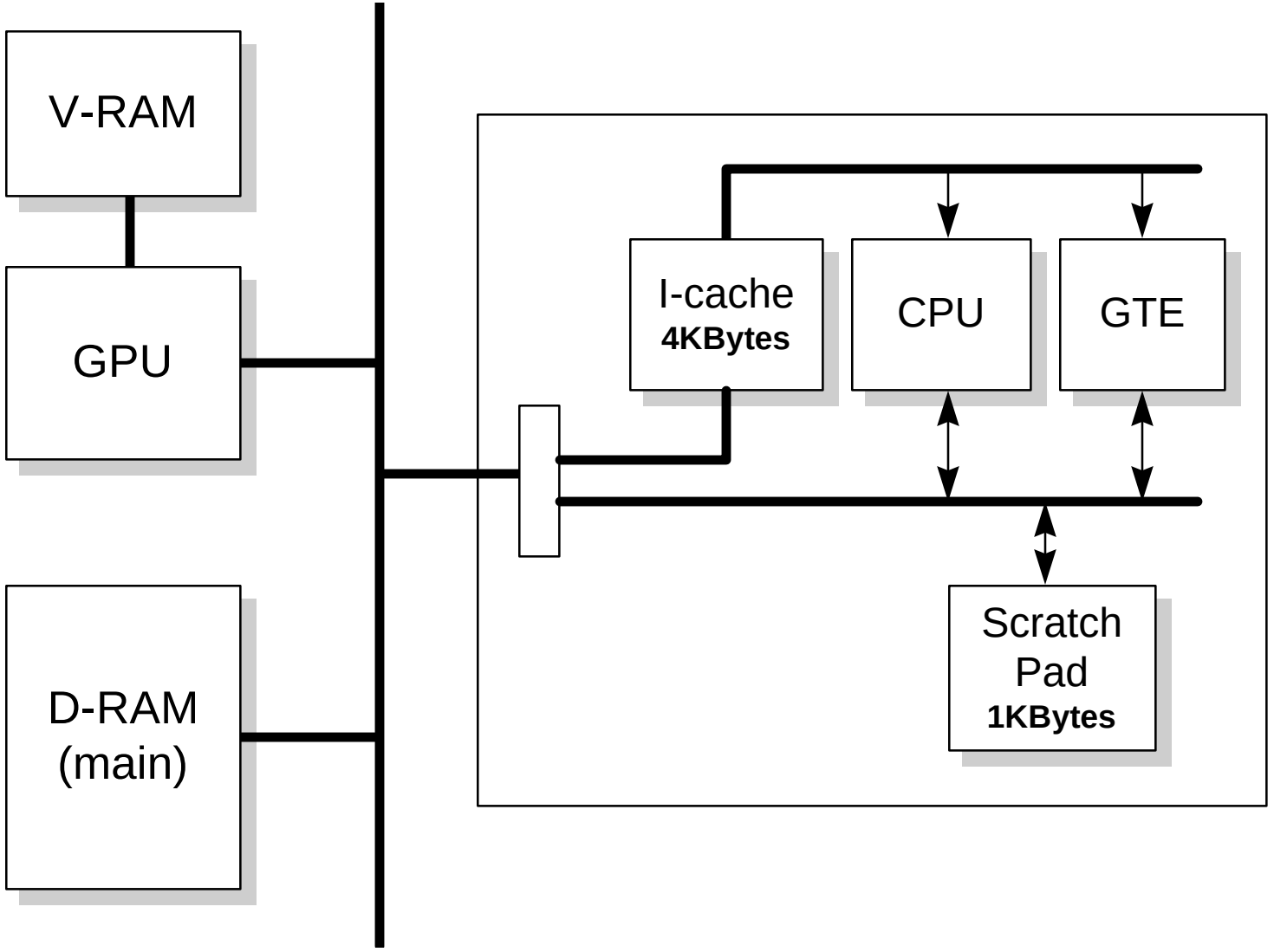
Model for improving speed

A. **On Cache**

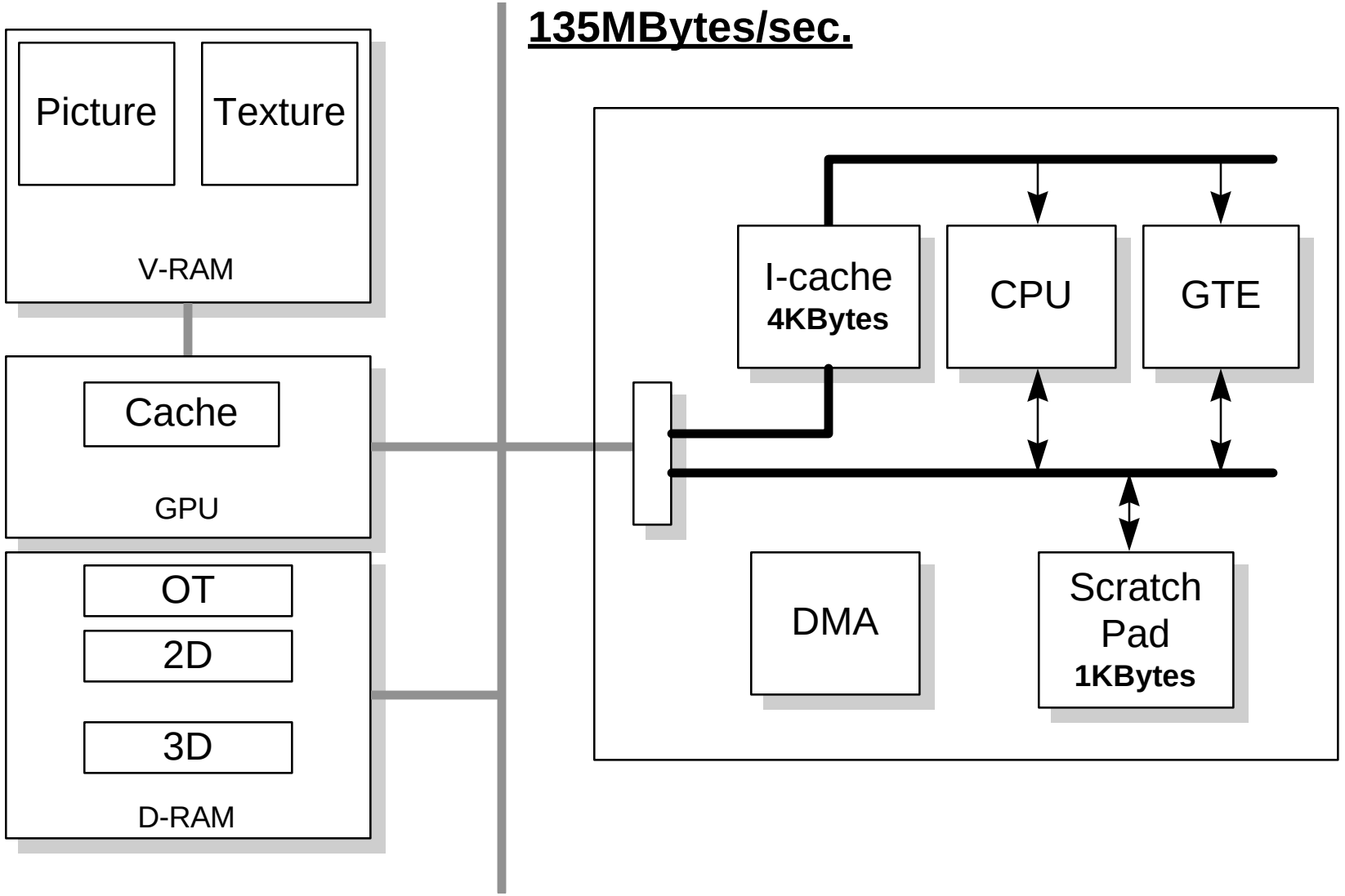
B. **DMPSX**

C. **Scratch Pad**

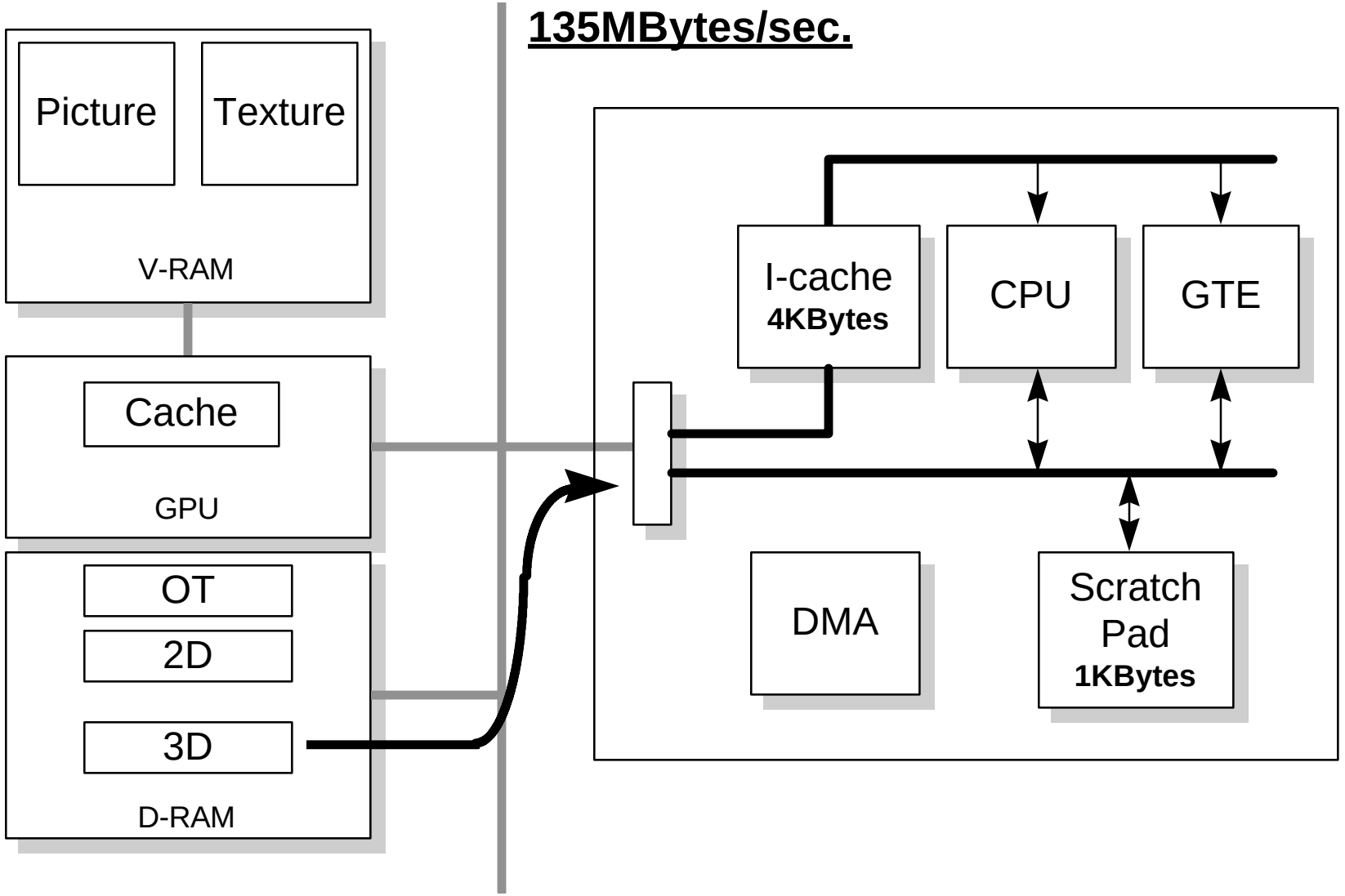
PlayStation architecture



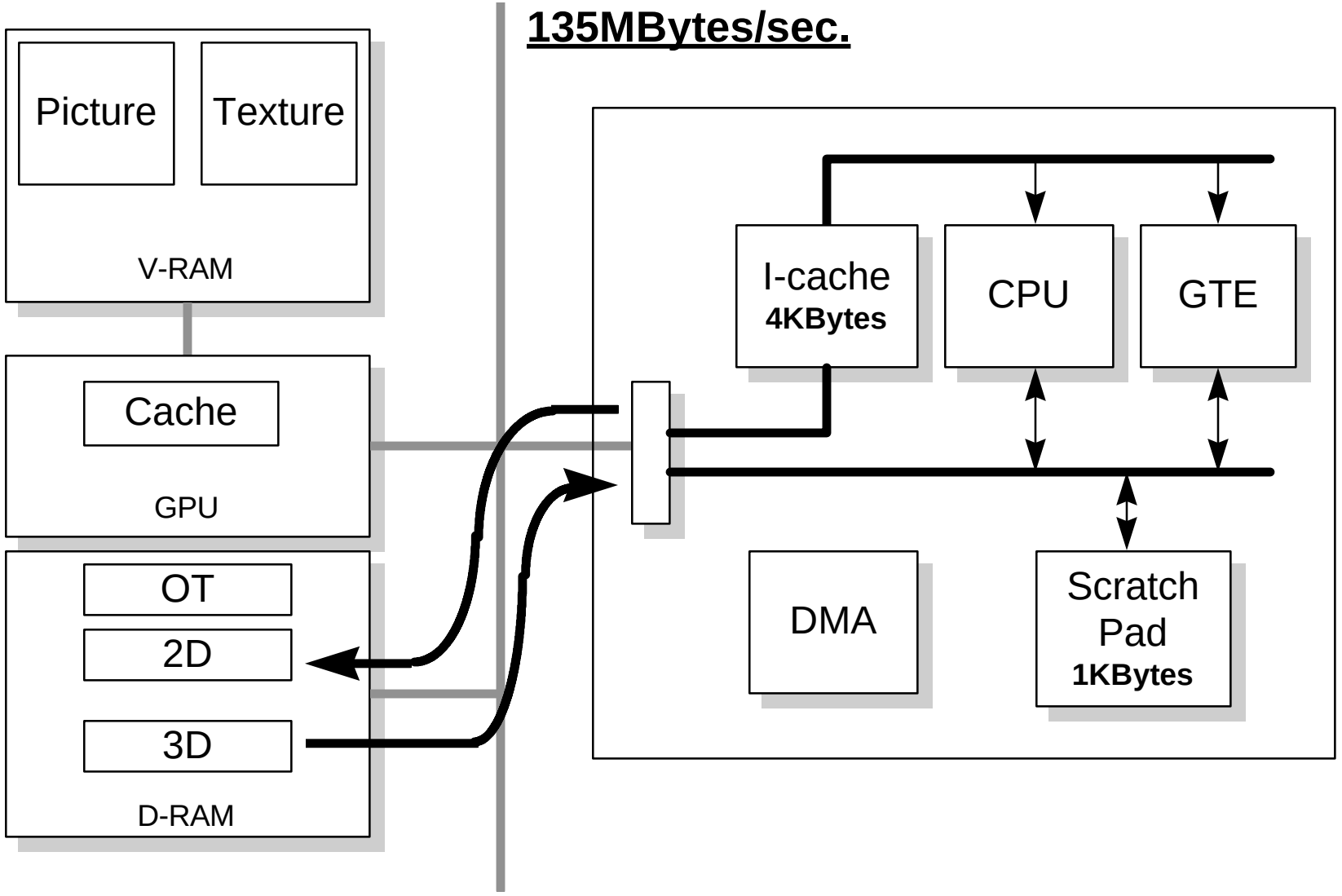
PlayStation architecture



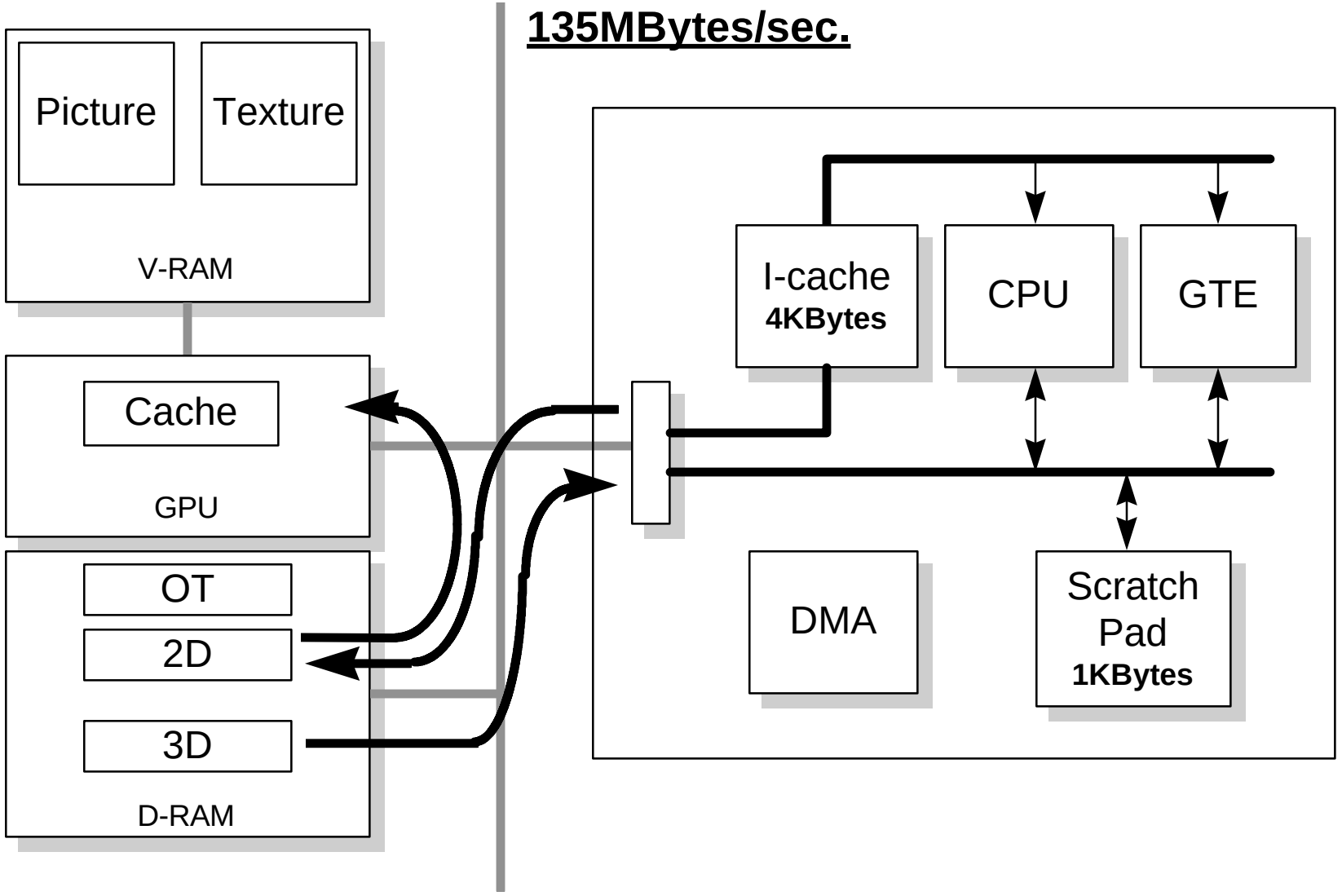
PlayStation architecture



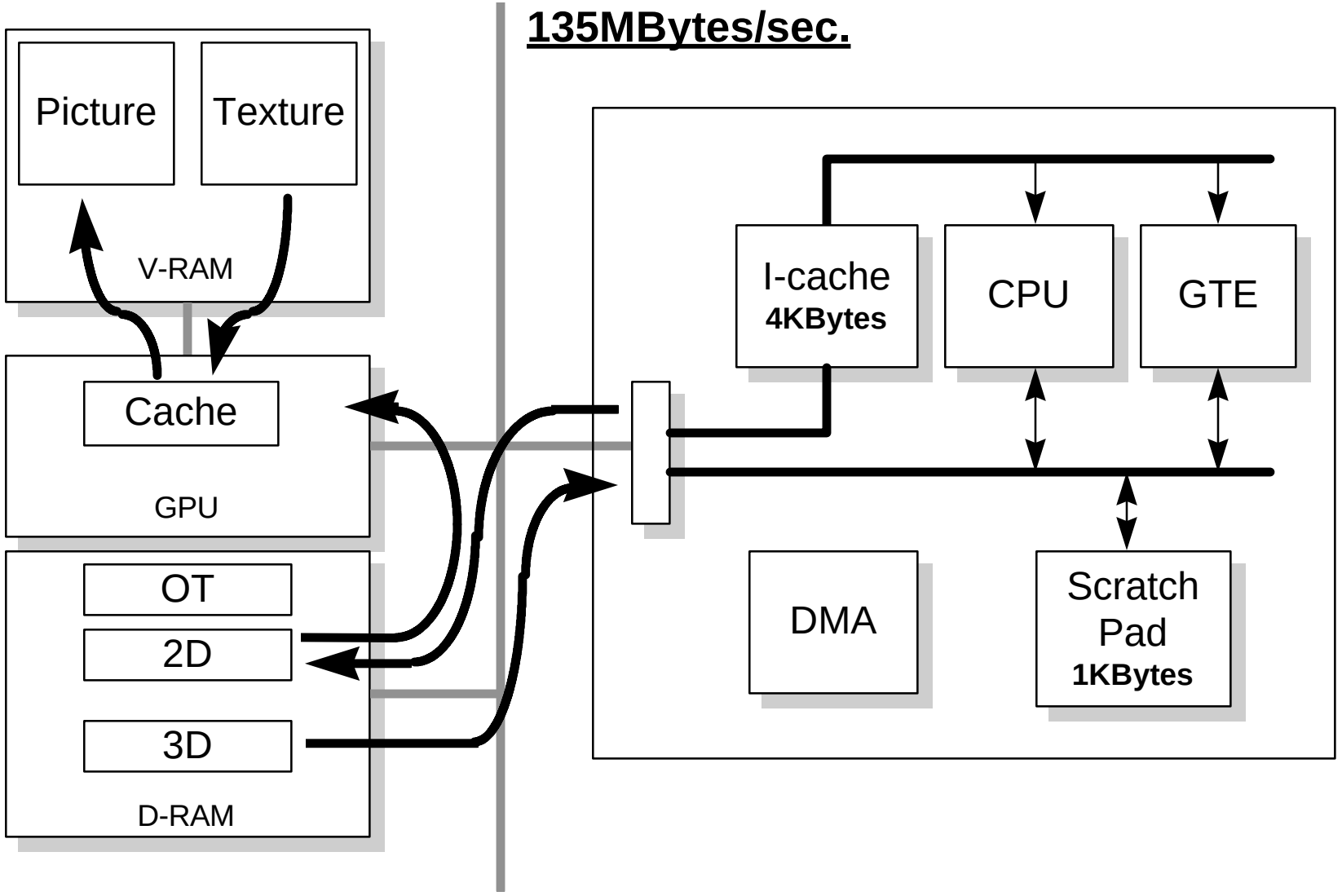
PlayStation architecture



PlayStation architecture



PlayStation architecture

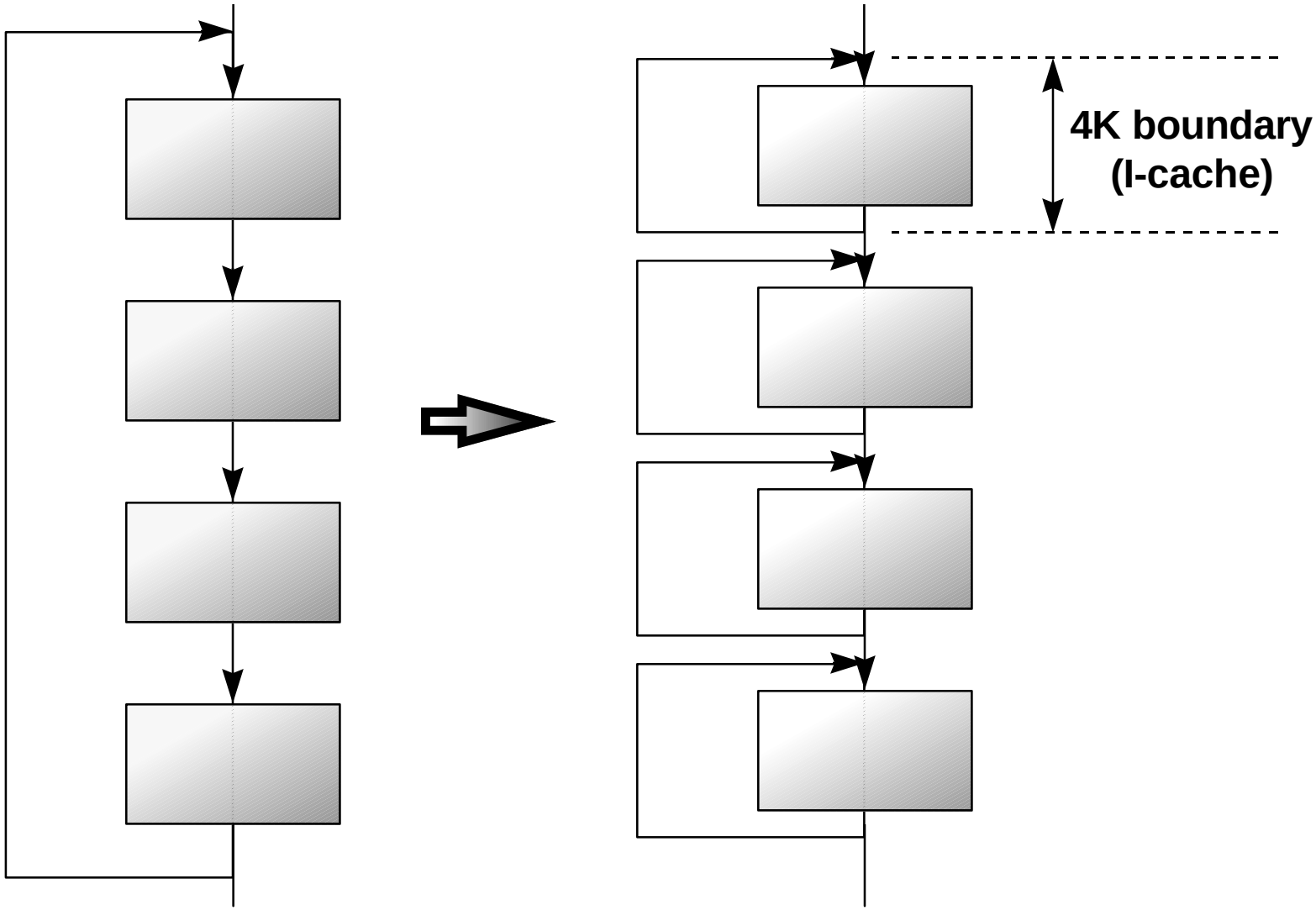


Always turn cache on

(ON CACHE)!

Write programs so that commands and data can take advantage of I-cache and Scratch Pad

Using short loops

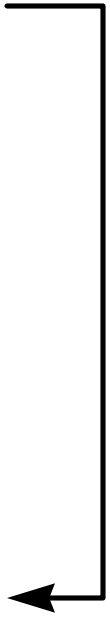


OFF CACHE using library (1)

```
func(){
   for(i = 0; i < n; i++){
      ........
      RotTransPers(V0, sxy, p, flag);
      ........
   }
}
```

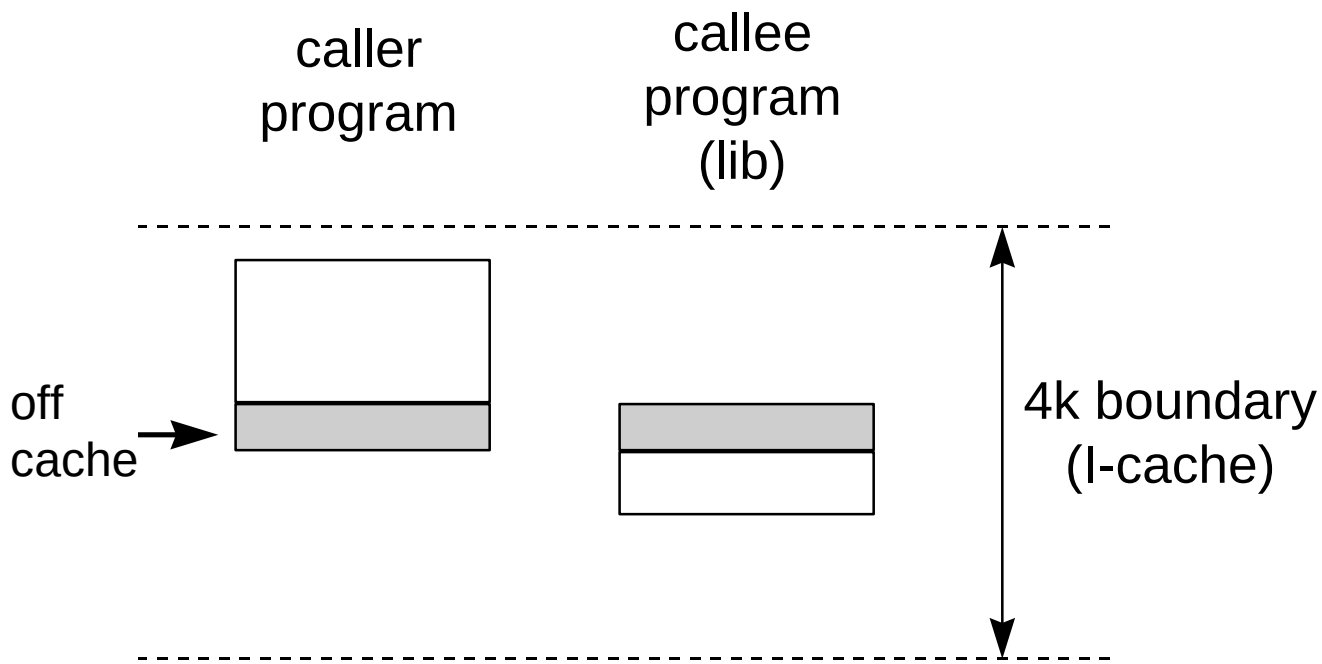
libgte:

```
      RotTransPers(V0, sxy, p, flag);
```



OFF CACHE using library (2)

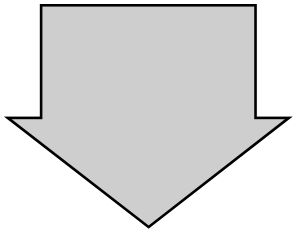
The PlayStation I-cache uses direct mapping so in cases such as the one shown below, small programs are OFF CACHE as well



What is DMPSX?

DMPSX

- a utility used when compiling a program
- directly inlines code where a function is called



This makes it possible to actively control how the program will operate in the I-cache.

DMPSX Configuration

GTEMAC.H: **libgte** described on DMPSX Basic Function

```
#define gte_RotTransPers(r1, r2, r3, r4, r5){\
        gte_ldv0(r1);               \
        gte_rtps():\
        gte_stsxy(r2);\
        gte_stdp(r3);\
        gte_stflg(r4);\
        gte_stszotz(r5);\
}
```

INLINE.H: Basic Function dummy cord

```
#define gte_rtps(){\
        __asm__volatile(".word 0x00000a3f":::"$12","$13","$14","$15","memory");\
        __asm__volatile(".word 0x00000a3e":::"$12","$13","$14","$15","memory");\
        __asm__volatile(".word 0x00000a3e":::"$12","$13","$14","$15","memory");\
}
```

DMPSX.EXE: Utility for converting a dummy cord to a real cord (DMPSX mainframe)

Improving speed using DMPSX (inline compiler) (1)

```
func(){
   for(i = 0; i < n; i++){
      ........
      RotTransPers(V0, sxy, p, flag);
      ........
   }
}
```

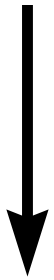


```
func(){
   for(i = 0; i < n; i++){
      ........
      gte_RotTransPers(V0, sxy, p, flag, otz);
      ........
   }
}
```

Always ON CACHE

Improving speed using DMPSX (inline compiler) (2)

```
*otz = RotTransPers(v0, sxy, p, flag);
```



```
gte_RotTransPers(v0, sxy, p, flag, otz);
```

Valid when the command is OFF CACHE

Improving speed using DMPSX (inline compiler) (3)

Contents of Inline Functions

```
gte_RotTransPers(v0, sxy, p, flag, otz);
```

||

```
{
  gte_ldv0(v0);  /* type1: load 3D coordinate */
  gte_rtps();    /* type2: Rotate, Transfer, Perspect */
  gte_stsxy(sxy);          /* type3: store 2D coordinate */
  gte_stdp(p);   /* type3: store depth que p */
  gte_stflg(flag);/* type3: store flag */
  gte_stszotz(otz);        /* type3: store sz/4 as otz */
}
```

: see gtemac.h

type1: “Register Load Functions”
type2: “GTE Commands”
type3: “Register Store Functions”

Improving speed using DMPSX (inline compiler) (4)

Program using Basic Functions

```
{
  gte_ldv0(v0);       /* type1: load 3D coordinate */
  gte_rtps();         /* type2: Rotate, Transfer, Perspect */
  gte_stsxy(sxy);     /* type3: store 2D coordinate */
  gte_stdp(p);        /* type3: store depth que p */
  gte_stflg(flag);    /* type3: store flag */
  gte_stszotz(otz);   /* type3: store sz/4 as otz */
}
```

Elimination of unnecessary commands

```
{
  gte_ldv0(v0);       /* type1: load 3D coordinate */
  gte_rtps();         /* type2: Rotate, Transfer, Perspect */
  gte_stsxy(sxy);     /* type3: store 2D coordinate */
  gte_stdp(p);        /* type3: store depth que p */
  gte_stflg(flag);    /* type3: store flag */
  gte_stszotz(otz);   /* type3: store sz/4 as otz */
}
```

Improving speed using DMPSX (inline compiler) (5)

Program using Basic Functions

```
{
   gte_ldv0(v0);
   gte_rtps();
   gte_stsxy(sxy);
   gte_stdp(p);
   gte_stflg(flag);
   gte_stszotz(otz);
}
```

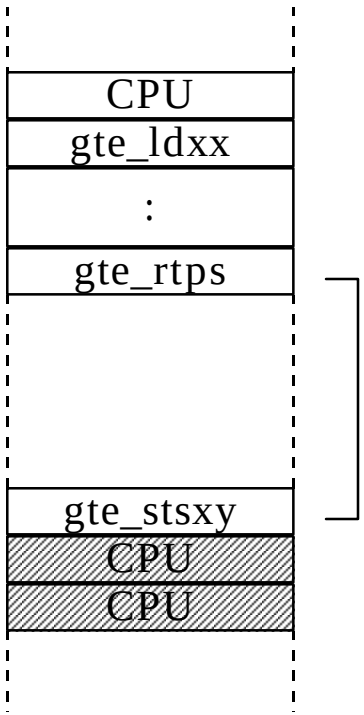
CPU process is inserted between GTE command (type2) and store command (type3)

In this operation, GTE and CPU are operating in parallel

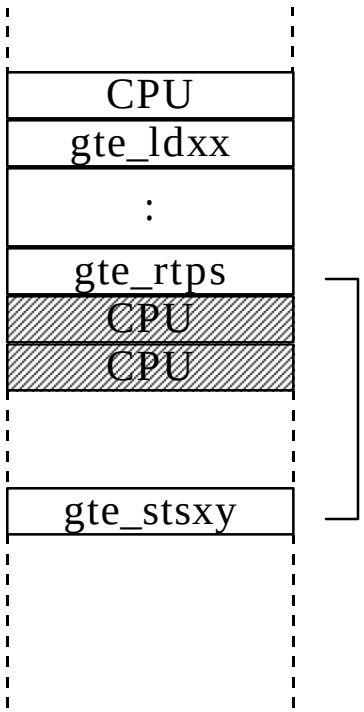
```
{
   gte_ldv0(v0);
   gte_rtps();
   gte_stsxy(sxy);
   CPU process;
   gte_stszotz(otz);
}
```

Improving speed using process reorder

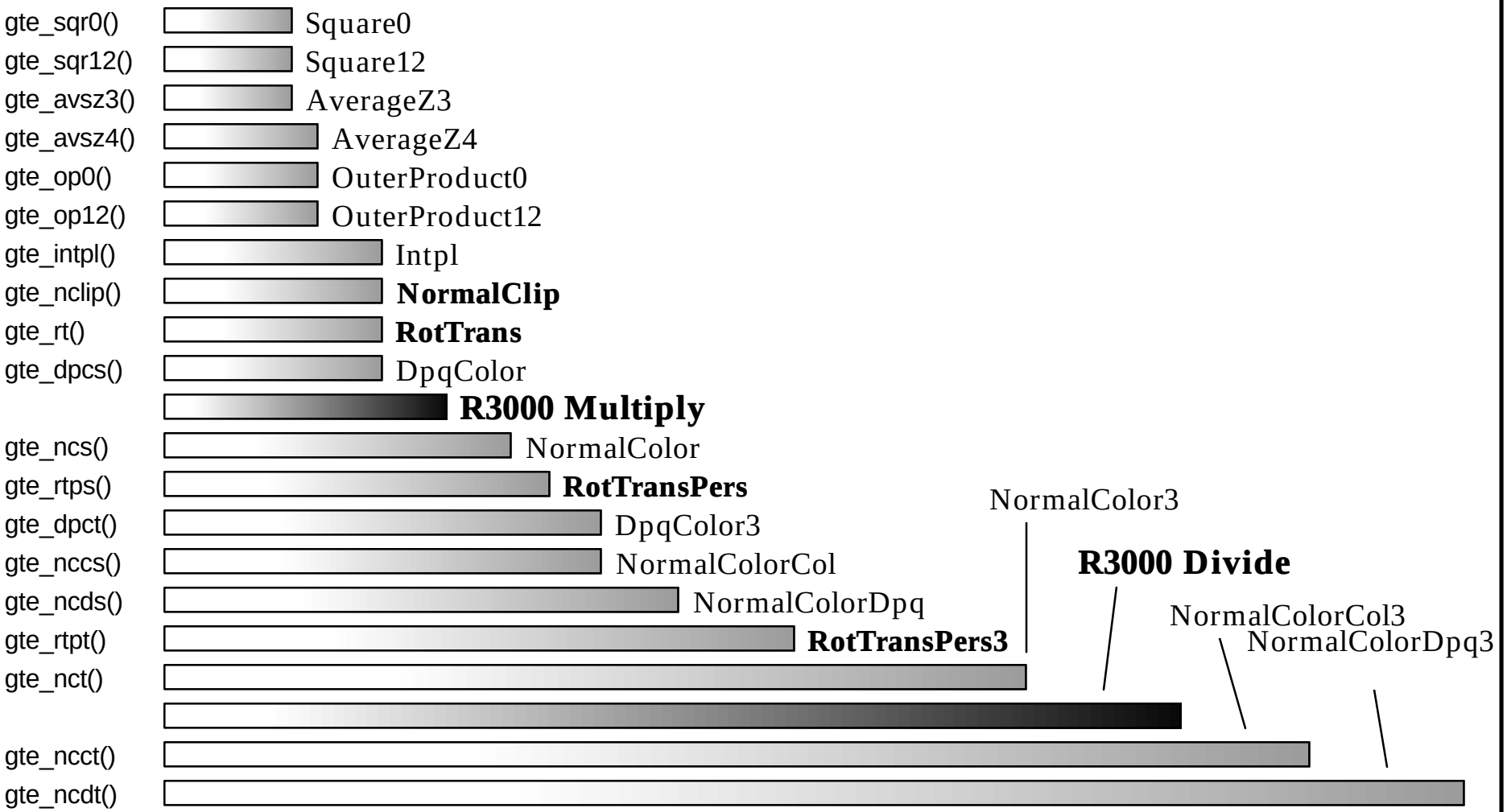
```
{
   gte_ldv0(v0);
   gte_rtps();
   gte_stsxy(sxy);
   CPU process;
}
```




```
{
   gte_ldv0(v0);
   gte_rtps();
   CPU process;
   gte_stsxy(sxy);
}
```



Calculation speed using GTE



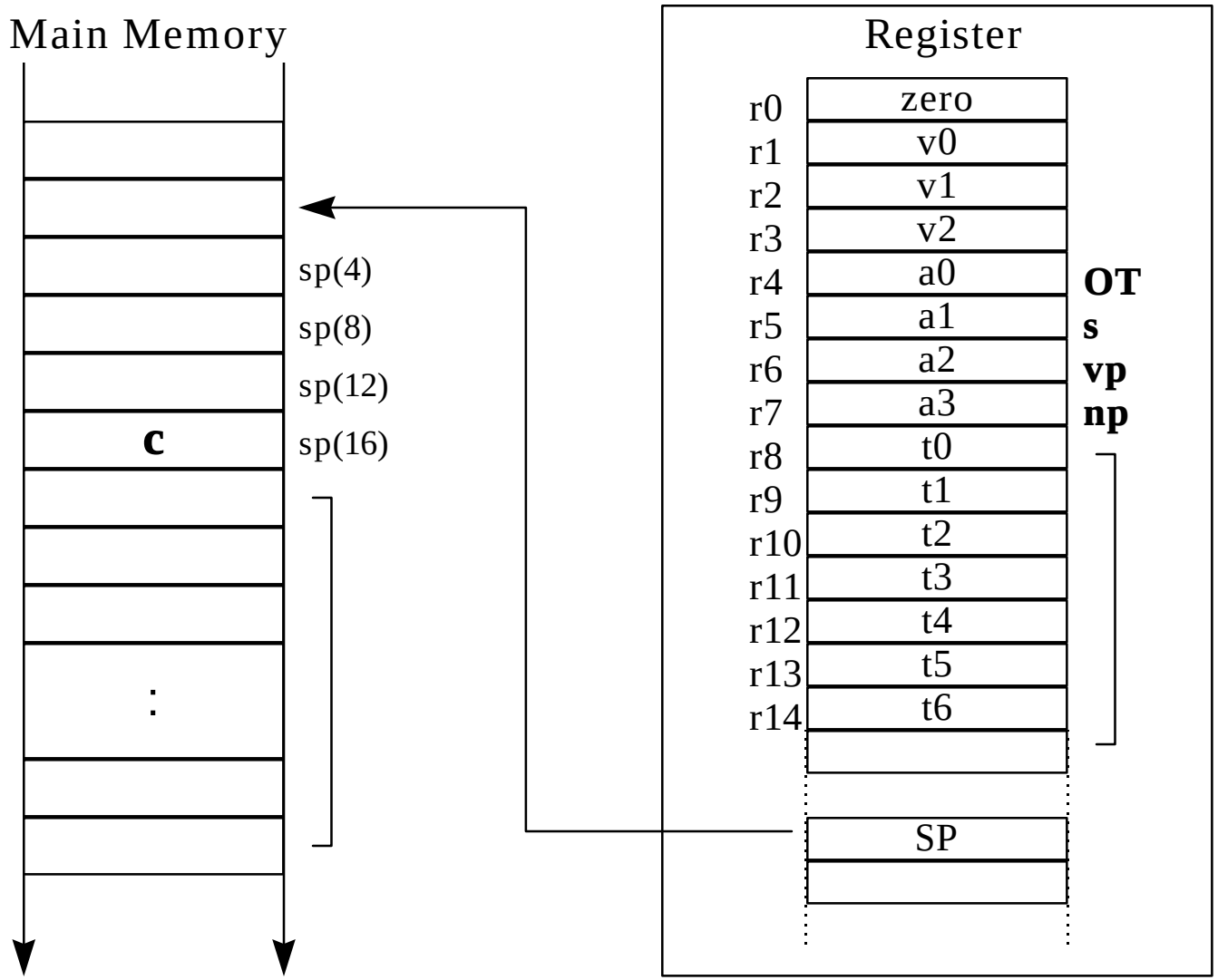
Sample program

```
add_cube(u_long *ot, POLY_G3 *s, SVECTOR **vp, SVECTOR **np, CVECTOR *c)
{
   int i;
   long otz, flg, clip;
   for(i = 0; i < 12; i ++, s ++, vp += 3, np += 3){
      clip = RotAverageNclipColorCol3(
         vp[0], vp[1], vp[2], np[0], np[1], np[2],
         &c[i],
         (long *)&s->x0, (long *)&s->x1, (long *)&s->x2,
         (CVECTOR)&s->r0, (CVECTOR)&s->r1, (CVECTOR)&s->r2,
         &otz, &flg);
      if(clip <= 0) continue;
      if((flg & 0x80000000) == 0){
         otz >>= (14-OTLENGTH);
         addPrim(ot+OTSIZE-otz, s);
      }
   }
}
```

Improving speed using scratch pad (1)

```
add_cube(u_long *ot, POLY_G3 *s, SVECTOR *vp, SVECTOR **np, CVECTOR *c)
{
   int i;
   struct wk {
      u_long *ot;
      long otz, flg, clip;                          /* Local Work Area */
      CVECTOR *col;
   } *wk;
   wk = (struct wk *)getScratchAddr(0);             /* set scratch pad address */
   wk->col = c;
   wk->ot = ot;
   for(i = 0; i < 12; i ++, s ++, vp += 3, np += 3){
      wk->clip = RotAverageNclipColorCol3(
         vp[0], vp[1], vp[2], np[0], np[1], np[2],
         &wk->col[i],
         (long *)&s->x0, (long *)&s->x1, (long *)&s->x2,
         (CVECTOR)&s->r0, (CVECTOR)&s->r1, (CVECTOR)&s->r2,
         &otz, &flg);
      if(wk->clip <= 0) continue;
      if((wk->flg & 0x80000000) == 0){
         wk->otz >>= (14-OTLENGTH);
         addPrim(wk->ot+OTSIZE-wk->otz, s);
      }
   }
}
```

Mapping of arguments to C compiler



Improving speed using scratch pad (2)

```
typedef struct {
   u_long *ot;
   long otz, flg, clip;                    /* Local Work Area */
   CVECTOR *c;
} WK;

add_cube(WK *wk, POLY_G3 *s, SVECTOR **vp, SVECTOR **np)
{
   int i;
   for(i = 0; i < 12; i ++, s ++, vp += 3, np += 3){
      wk->clip = RotAverageNclipColorCol3(
         vp[0], vp[1], vp[2], np[0], np[1], np[2],
         &wk->col[i],
         (long *)&s->x0, (long *)&s->x1, (long *)&s->x2,
         (CVECTOR)&s->r0, (CVECTOR)&s->r1, (CVECTOR)&s->r2,
         &otz, &flg);
      if(wk->clip <= 0) continue;
      if((wk->flg & 0x80000000) == 0){
         wk->otz >>= (14-OTLENGTH);
         addPrim(wk->ot+OTSIZE-wk->otz, s);
      }
   }
}
```

Improving speed using scratch pad and DMPSX

```
add_cube(WK *wk, POLY_G3 *s, SVECTOR **vp, SVECTOR **np)
{
  int i;
  for(i = 0; i < 12; i ++, s ++, vp += 3, np += 3){
    gte_ldv3(vp[0], vp[1], vp[2]);                  /* type1 */
    gte_rtpt(0);                                    /* type2 */
    gte_stflg(&wk->flg);                            /* type3 */
    gte_nclip();                                    /* type2 */
    gte_stopz(&wk->clip);                                     /* type3 */
    if(wk->clip <= 0) continue;
    gte_ldv3(np[0], np[1], np[2]);                  /* type1 */
    gte_ldrgb(&wk->c[i]);                                     /* type1 */
    gte_ncct();                                     /* type2 */
    if((wk->flg & 0x80000000) == 0){
      gte_stsxy3(&s->x0, &s->x1, &s->x2);           /* type3 */
      gte_strgb3(&s->r0, &s->r1, &s->r2);           /* type3 */
      gte_avsz3();                                  /* type2 */
      gte_stotz(&wk->otz);                          /* type3 */
      wk->otz >>= (14-OTLENGTH);
      addPrim(wk->ot+OTSIZE-wk->otz, s);
    }
  }
}
```

Scratch pad stack (1)

Scratch Pad is allocated to stack

- A maximum of 1KByte is allocated to stack

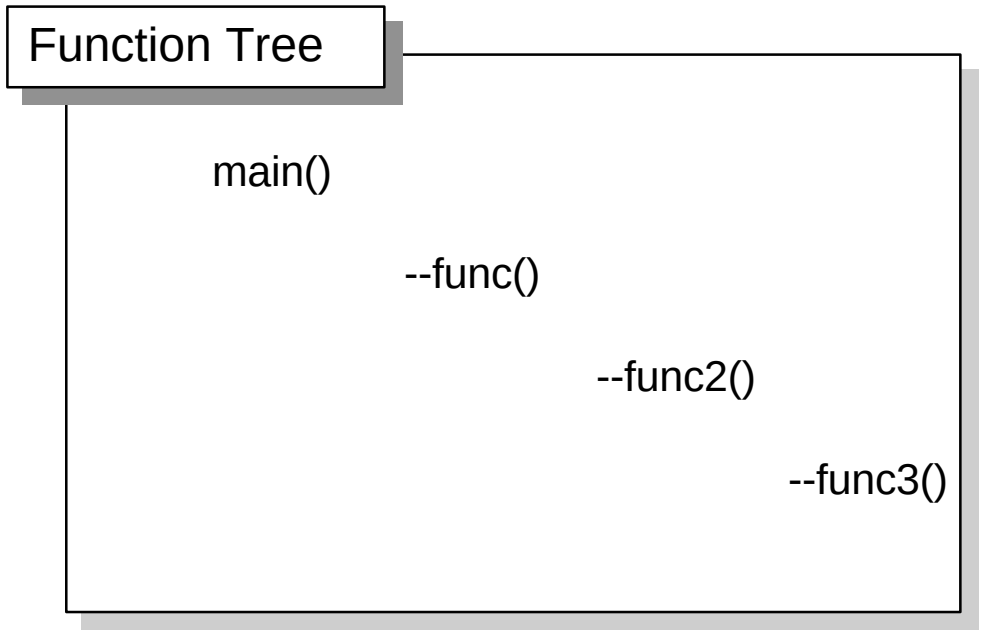
Scratch pad stack (2)

```
main()
{
   func1();
}

func1()
{
   func2();
}

func2()
{
   int i;

   for(i = 0; i < n; i ++) func3(i);
}
```



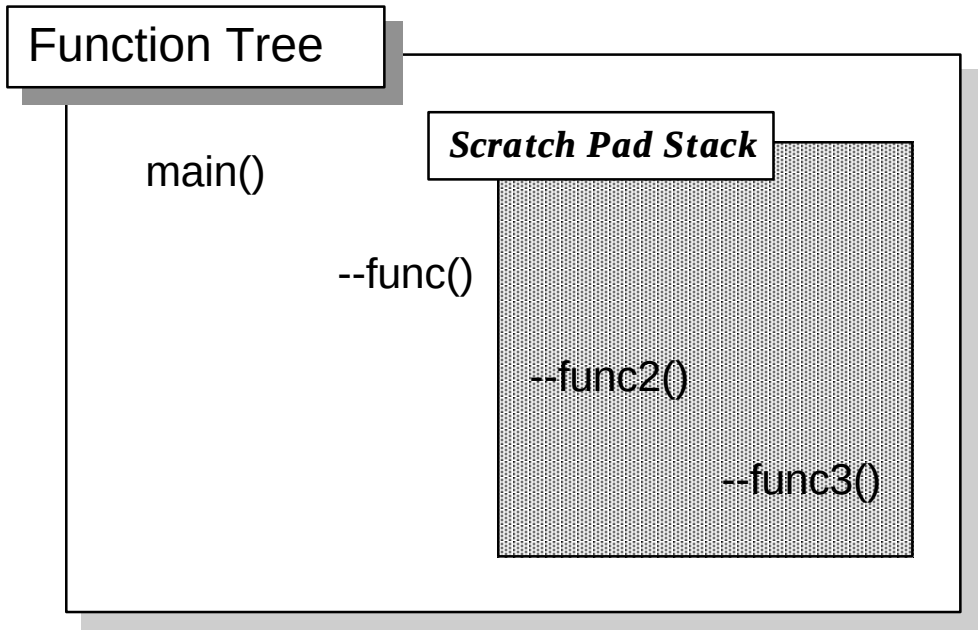
Scratch pad stack (3)

```
main()
{
   func1();
}

func1()
{
   SetSpadStack(0x1f8003fc);
   func2();
   ResetSpadStack();
}

func2()
{
   int i;

   for(i = 0; i < n; i ++) func3(i);
}
```



Improving speed summary

** POINT **

Decrease the use of Main RAM since it is 5-6 times slower than Scratch Pad

1. **Do not place local variables on the stack**
2. **Do not place arguments on the stack**

} Reduction of stack amount used

3. **Use Scratch Pad and DMPSX**
4. **Scratch Pad is allocated to stack**

Recursive-call programs

Advantages

- A relatively complex operation can be written as short code

This facilitates use of the I-cache, and makes it easy to write programs that do automatic division of polygons

Disadvantages

- Uses a large amount of stack space and is slow

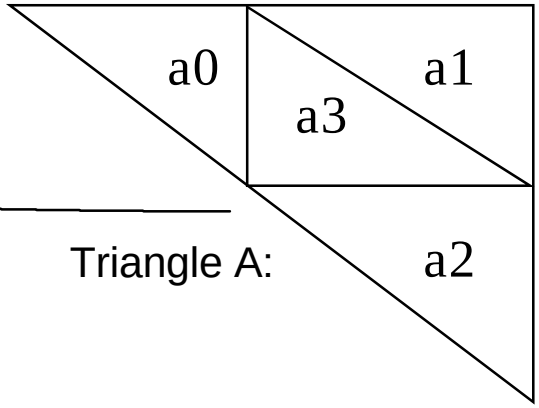
Increasing speed in recursive-call programs

- Decrease stack use
    1. Make a distinction between variables (arguments) that do not need to be placed on the stack and those that do
    2. The variables and arguments that do not need to be placed on the stack should be placed as structures in the **Scratch Pad**
    3. The necessary variables and arguments should be arranged in the **Scratch Pad** according to the level of recursion

An example of a recursive-call program

A simple triangle division : tridiv

tridiv(ot, s, v0, v1, v2, c0, c1, c2, n)




```
if(n == 0){
```

If at the lowest level, render triangle A

```
}
else{
```

For each side, calculate the center point and the color of the center point

```
tridiv(ot, s, a0, n-1);
tridiv(ot, s, a1, n-1);
tridiv(ot, s, a2, n-1);
tridiv(ot, s, a3, n-1);
```

recursive call

```
}
```

Customizing libgs

Able to customize routines such as coordinate conversion, light source calculation, packet production, etc.

- Jump table version GsSortObject
- Customizing insignificant functions using DMPSX

GsSortObject4J

1. GsSortObject4 ⟶ GsSortObject4J

2. Set only the necessary insignificant functions in_GsFCALL (Memory Saving)

3. Describe the processes you would like to customize as insignificant functions on DMPSX

4. Register on _GsFCALL

Default operation of insignificant functions

(see. \ psx \ sample \ graphics \ tmdview \ tmdview4 \ lowlevel)

1. Coordinate conversion and perspective change of vertex data

2. Flag decision

3. Back Face Clip

4. Light source calculations from normal data and brightness value

5. Registration to OT

6. Apply 1-5 for similar types of polygons

Cautions during assembly

- Handling of delay slots
  - Insertion of dangerous commands into delay slot
  - Careless deletion of nop

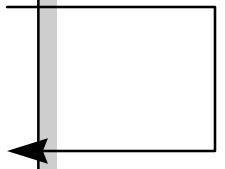
Be careful with programs that appear to work correctly on the surface

Example of dangerous cases 1 (dangerous command inserted into delay slot)

```
add v0, v0, t0
lw   t0, 0(a0)
nop
add v1, v1, t0
        :
        :
```

OK

```
. . . . . .
lw   t0, 0(a0)
add v0, v0, t0
add v1, v1, t0
        :
        :
```



NG!

Malfunction during normal interrupt

Example of dangerous cases 2 (careless deletion of nop)

```
      :
    beq v0, zero, L1
    nop
    lw   t0, 0(a0)
      :
      :
L1: add v0, v0, t0
    lw   t0, 4(a0)
      :
```

OK

```
      :
    beq v0, zero, L1
    lw   t0, 0(a0)
      :
      :
      :
L1: add v0, v0, t0
    lw   t0, 4(a0)
      :
```

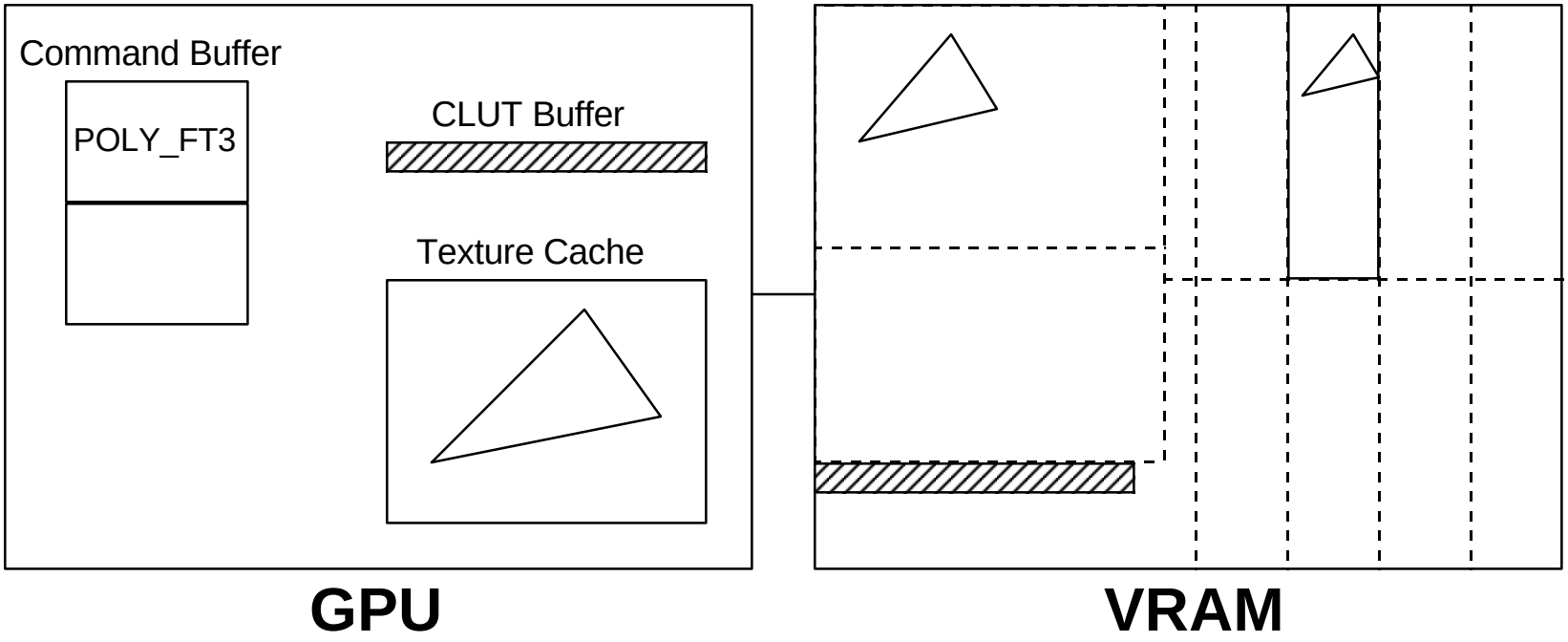
NG!

Malfunction during normal interrupt

Improving speed on GPU

- **Reduction of idle time**
- **Effective utilization of Texture Cache**

GPU Operation



- **16word** Command Buffer
- **2word x 256** Entry Texture Cache
- **128word** CLUT Buffer

Reduction of idle time

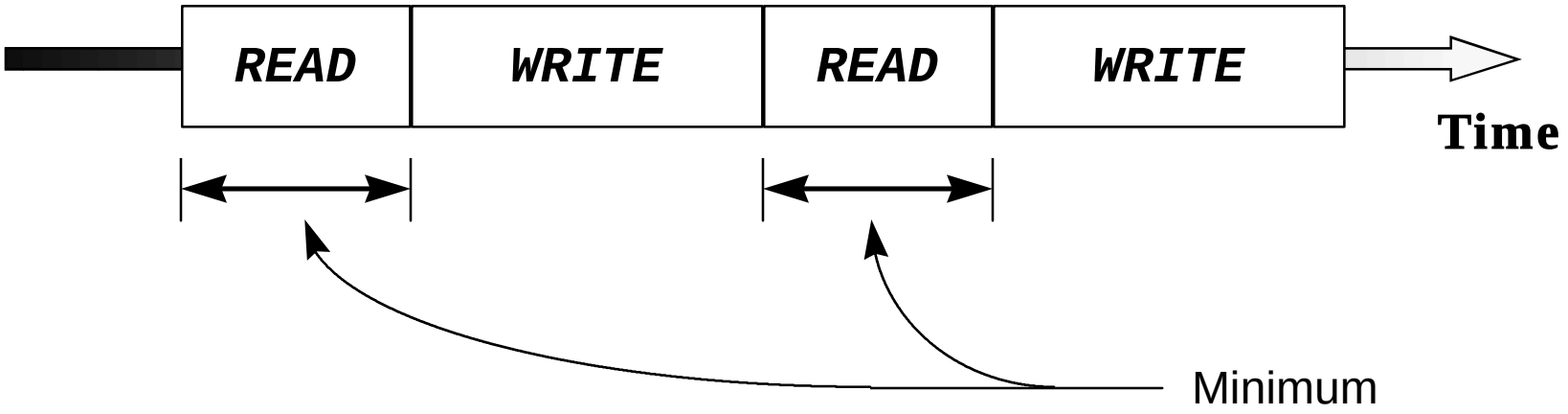
With Command Buffer is 1 Primitive at a time

NULL Primitive time is idle

Reduction of NULL Primitive = Reduction of idle time
(Clearing time of + OT is lowered)

Effective utilization of Texture Cache

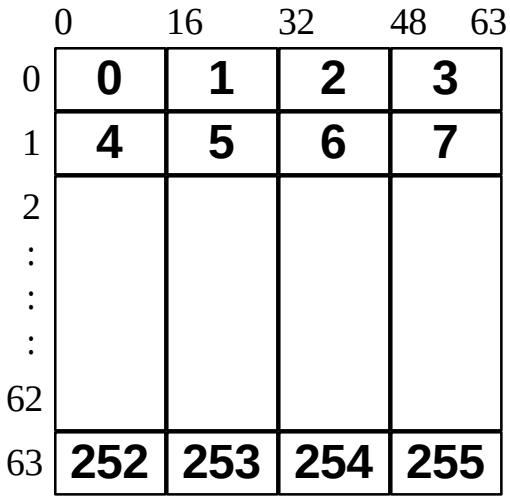
Texture Mapping



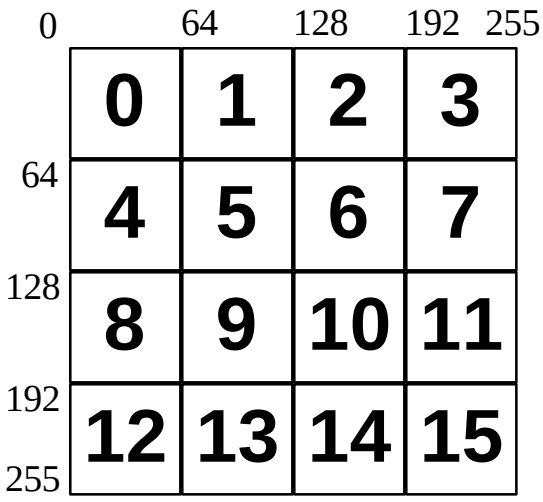
⇨ **Minimize Texture Lead Cycle**

Texture Cache

Mode	Total Size	1 Entry Size	Total Entry Number
4bit	64 x 64	16 x 1	256
8bit	64 x 32	8 x 1	256
16bit	32 x 32	4 x 1	256



Cache Entry



Cache Block

Texture Lead Cycle

Occurs in cache miss hit cases

- **Lowering of miss hit rate**
- **Decrease Lead Cycle**

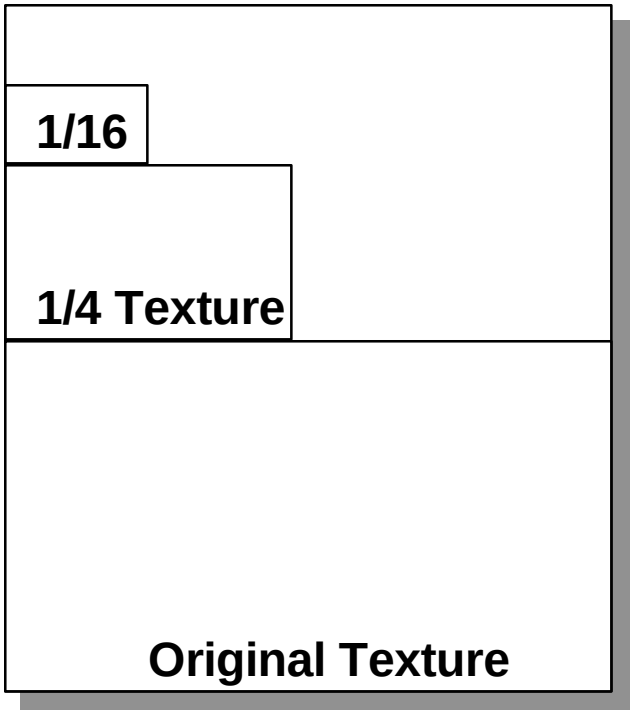
Reducing miss hit rate

- Make texture size less than cache size
- Use 4bit texture - maximum size of one entry
- Do not use reduction texture - Increase amount of available data in one entry

Reducing Lead Cycle

- Place the length texture horizontally
  - Lead is Y direction access time overhead
- Do not use reduction texture
  - Useless lead cycle occurs

Mip Mapping



In response to the polygon size the texture which was prepared beforehand should be properly used

Methods for speeding up polygon division

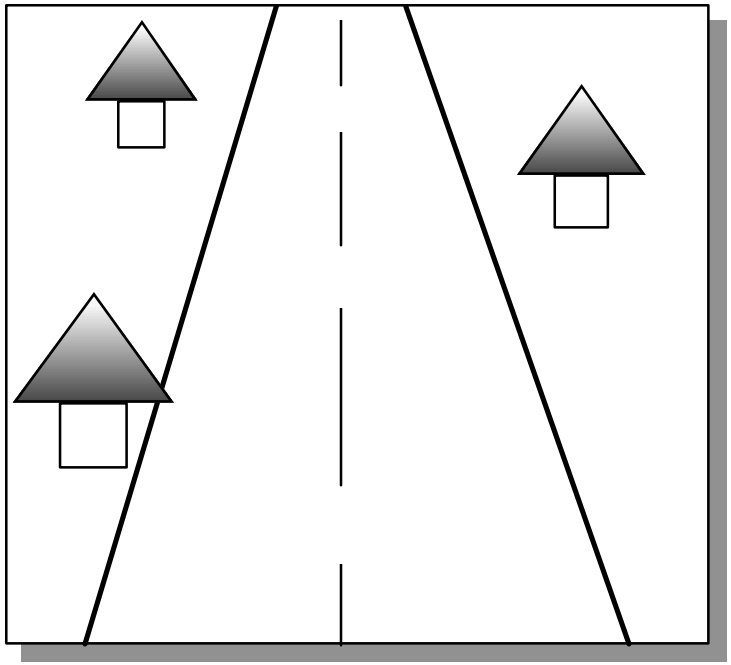
Displaying ground

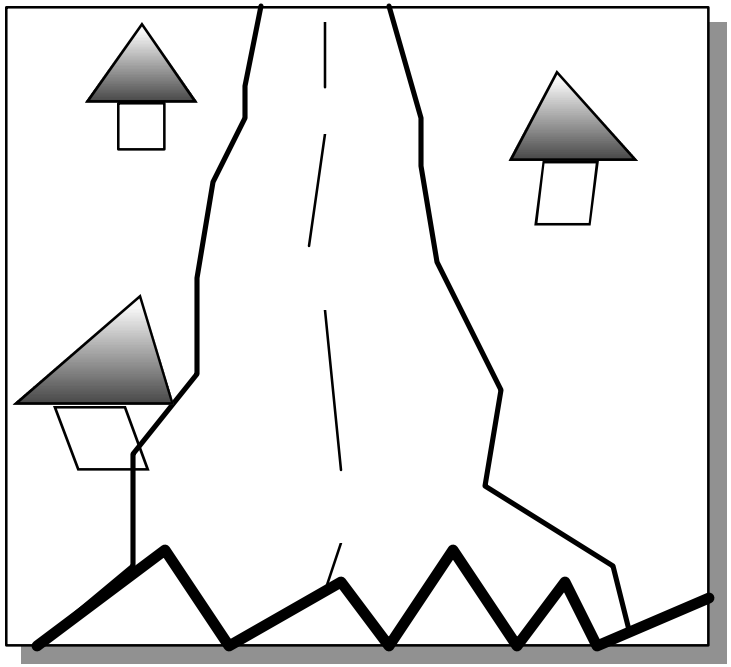
1. Determine method

2. Implementation

3. Improving speed

Problems involved in displaying ground

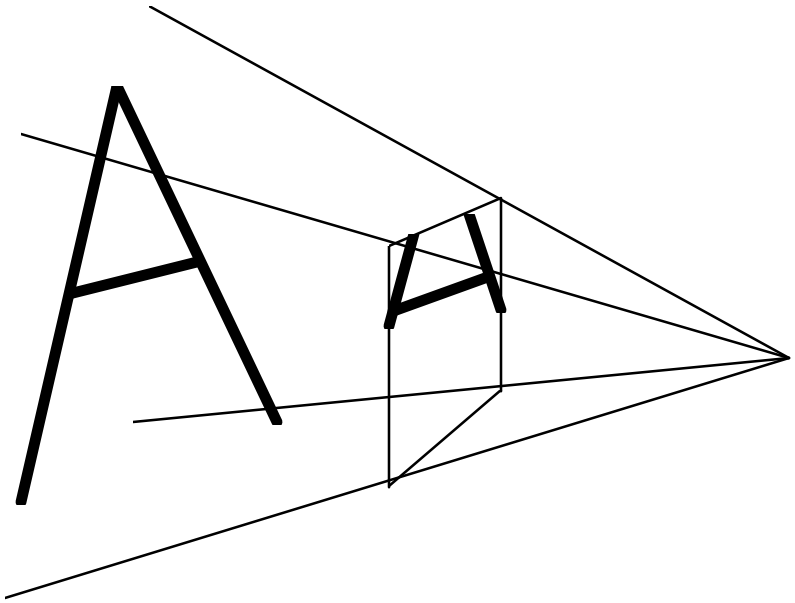




1. Warping of texture

2. Near clipping problems

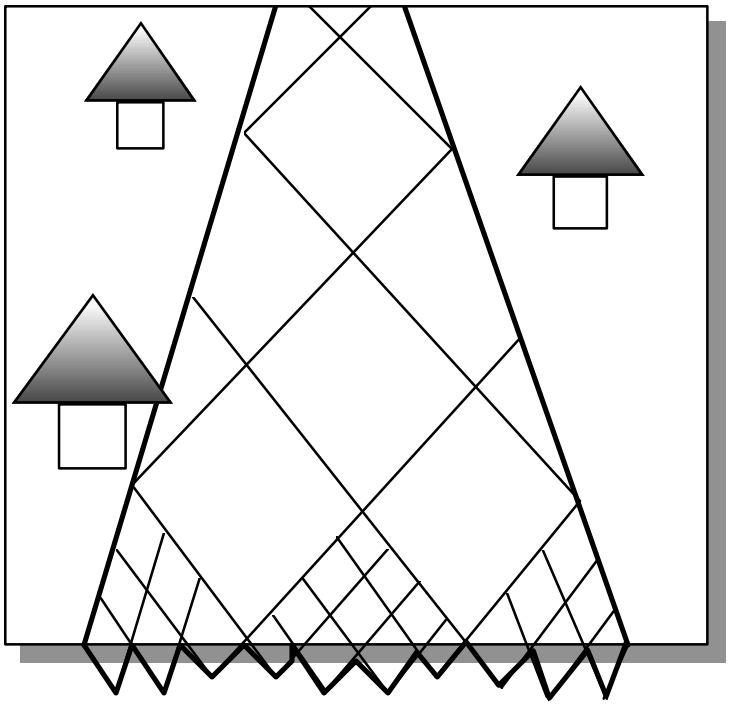
Solution using cone clipping



O allows more polygons to be used

X texture jumping
X texture warping
X calculations become more complex

Solution using division

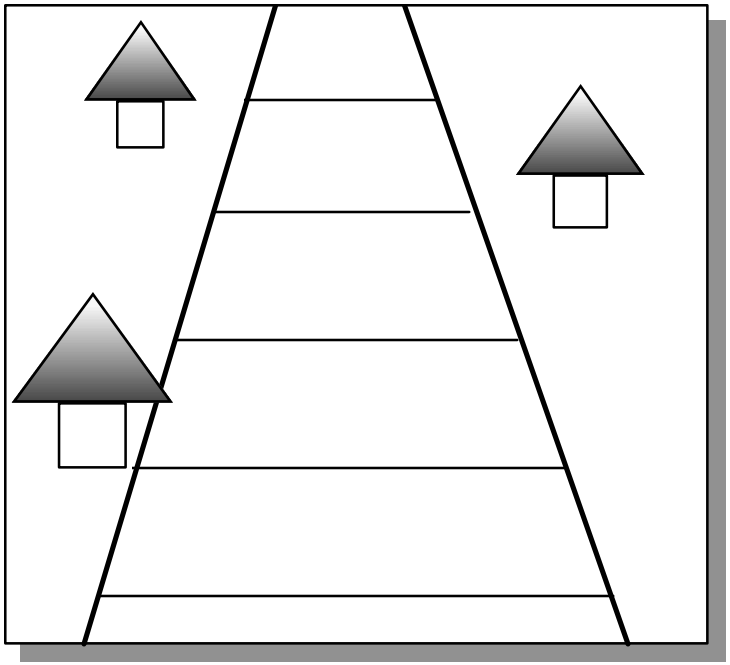


O less texture jumping
O texture warping is eliminated

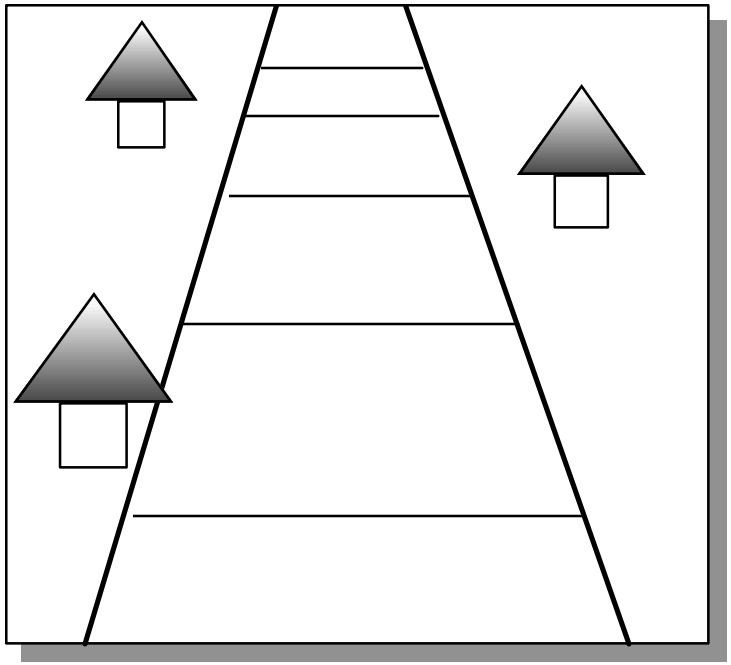
X the polygon count is increased

Using the division method is better!

Divide in 2 dimensions or 3 dimensions?



2 dimensions

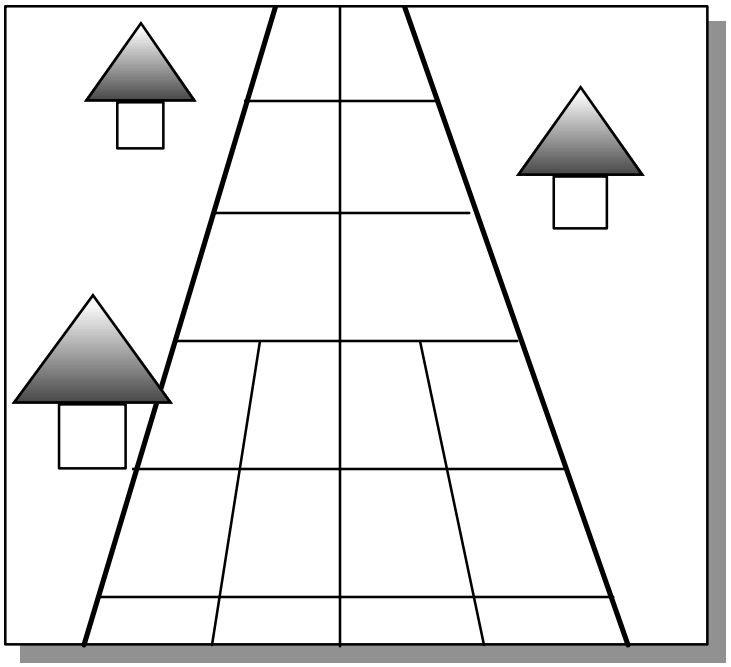


3 dimensions

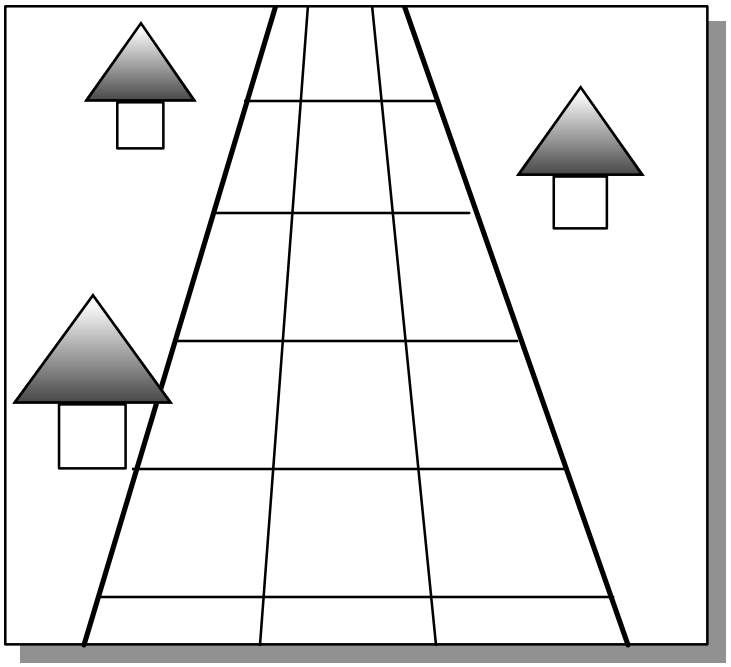
- 3 dimensions provides more accuracy
- Because GTE calculations are performed at high speeds, there is no overhead with 3-dimensional division

Divide in three dimensions

Active division or fixed division?



Active



Fixed

Use active method

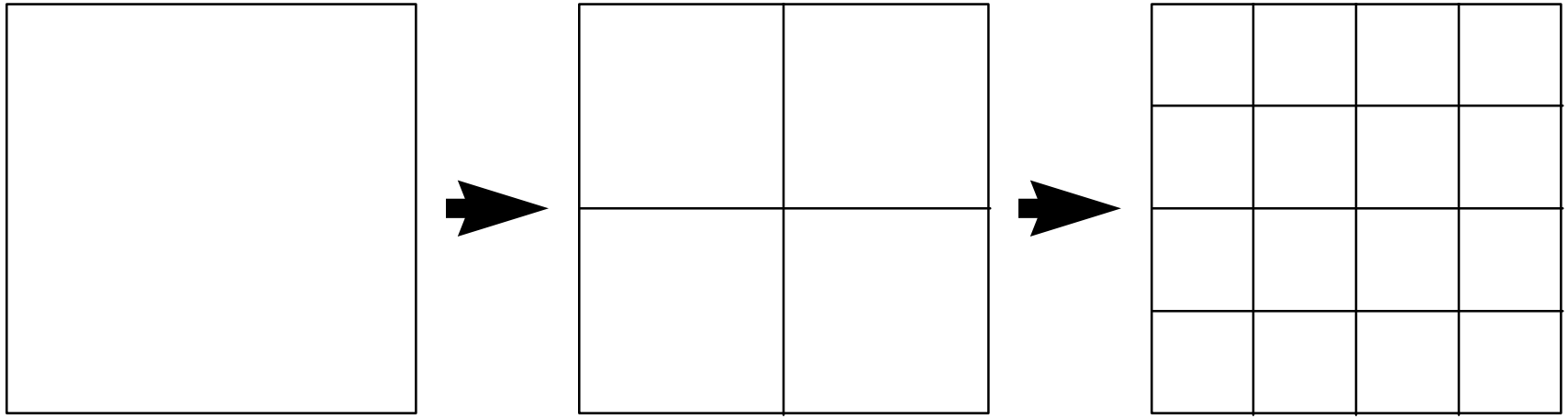
Advantages	Disadvantages
1. Polygon count is decreased 2. Improves speed	1. Gaps are generated 2. Textures become non- continuous

Actual programming

Principle

Display ground using active, 3-dimensional division

Recursive call



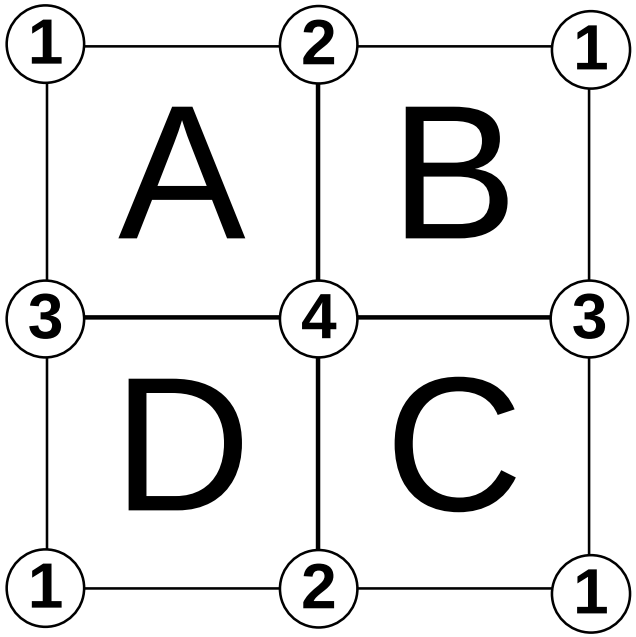
2^n division

<Polygon vertex distance>

Reasons

- GPU rendering limit 1024x512
- Polygon warping is most noticeable with larger polygons
- Used together with Area Clipping

3-Dimensional 2^n division

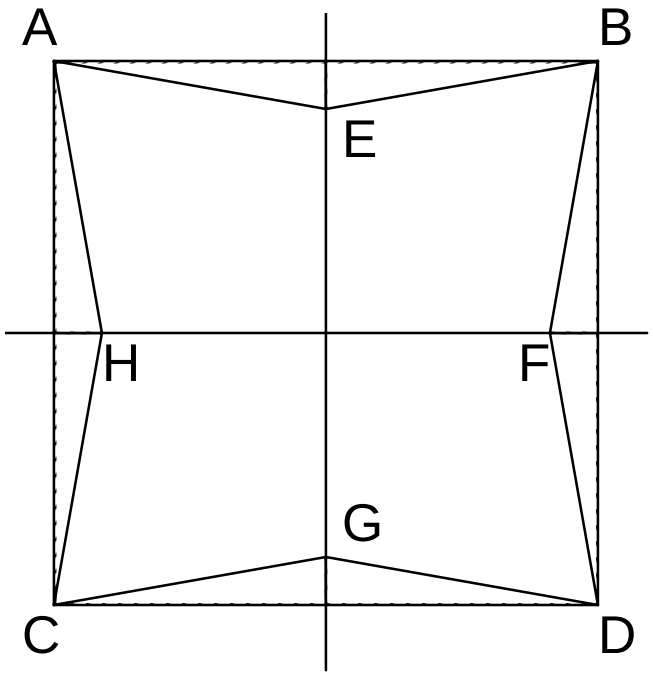


ordered as follows: A->B->C->D

x, y, z	coordinates
r, g, b	color
u, v	texture

These are all divided by two

Fixing gaps



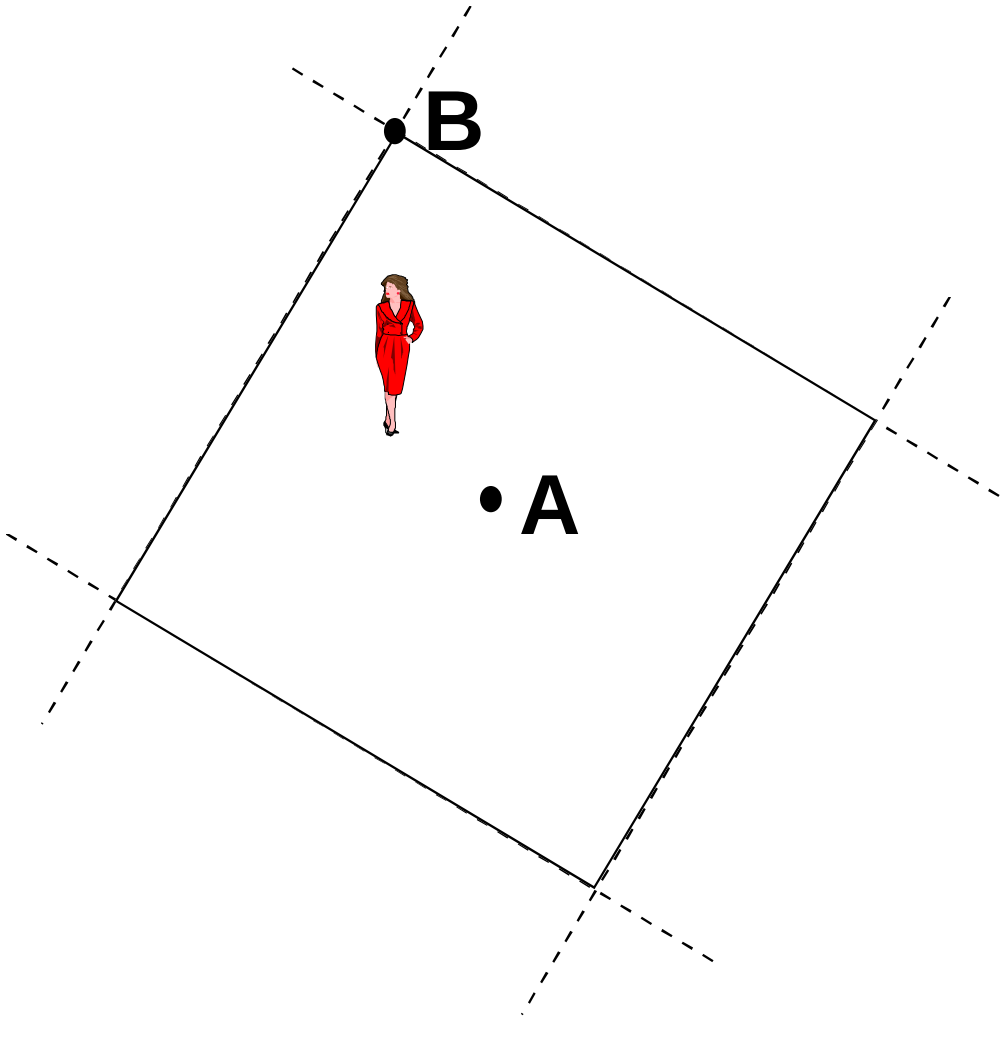
Reason

Due to the margin of error, the center point does not necessary lie on the axis

Solution

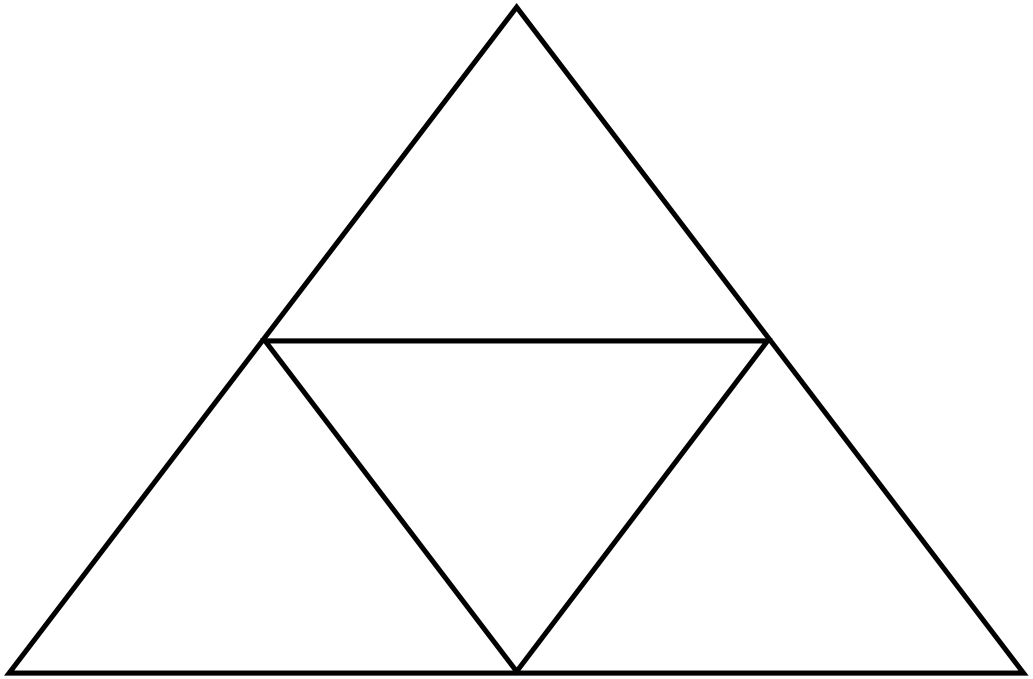
Draw a triangle for the gap as well
However, Back Clip is necessary

Solving the Z-sort problem



Set the Z-sort point to the furthest point (B) rather than the center of gravity (A)

Division of three-sided polygon



Always turn cache on (ON CACHE)!

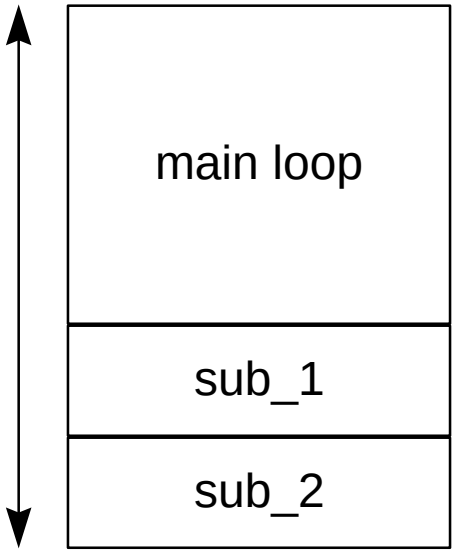
(for calculations of the polygon unit)

Instruction cache (1)

c:\> dumpsym main.sym | sort

Lower 12 bits or less

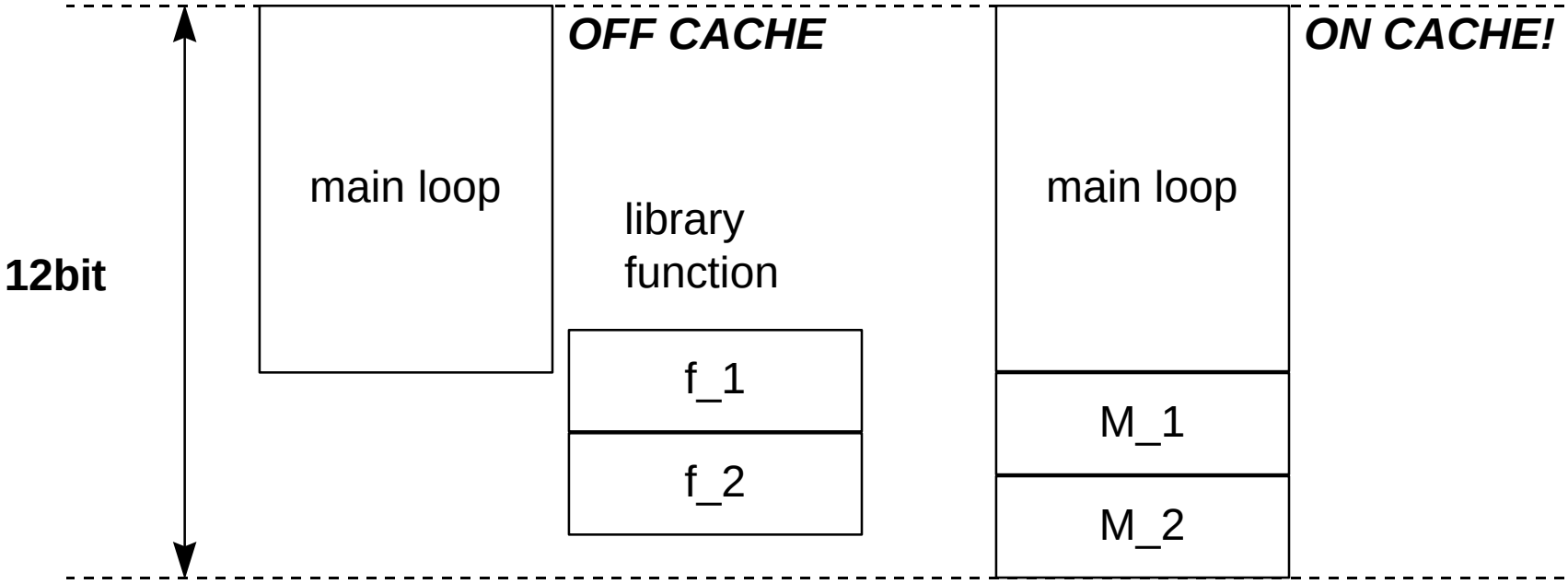
12bit or less



needs to be completed

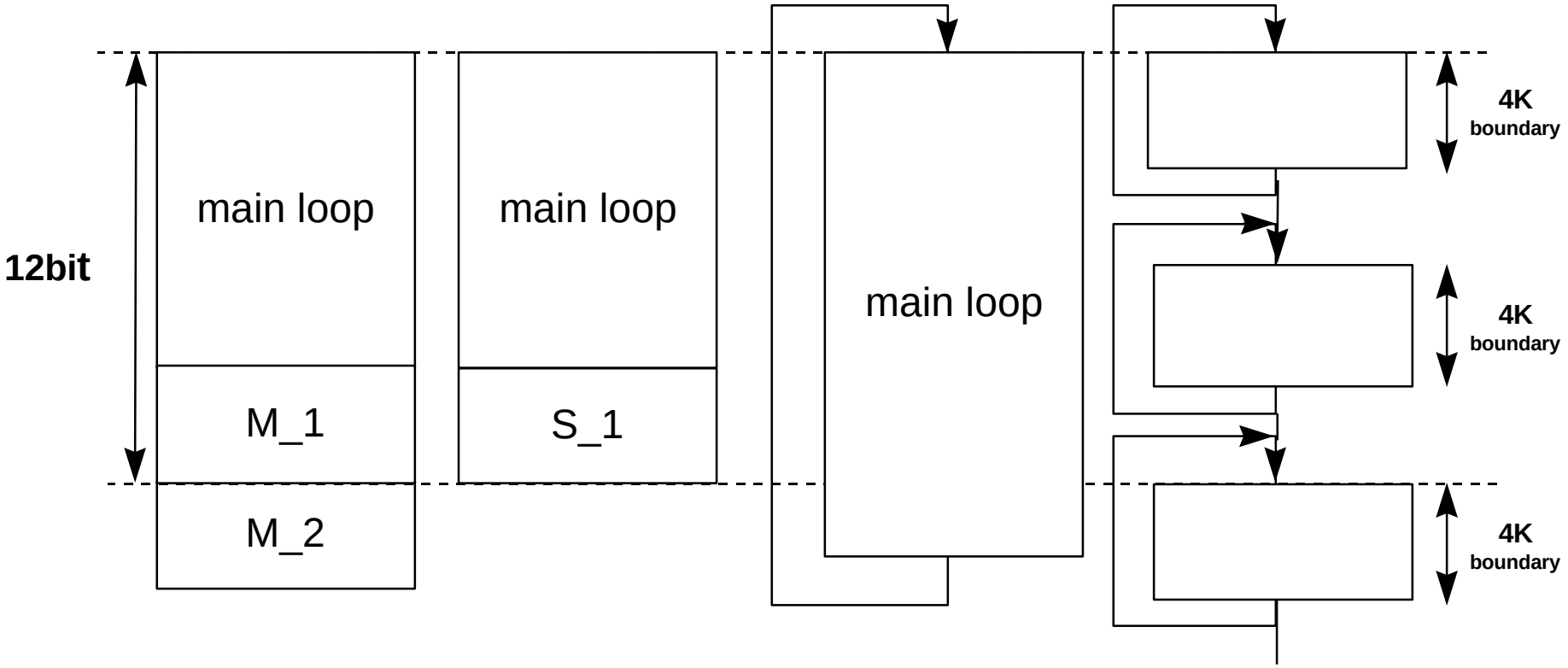
Instruction cache (2)

Cannot call library functions
Use library macros (**dmpsx**)

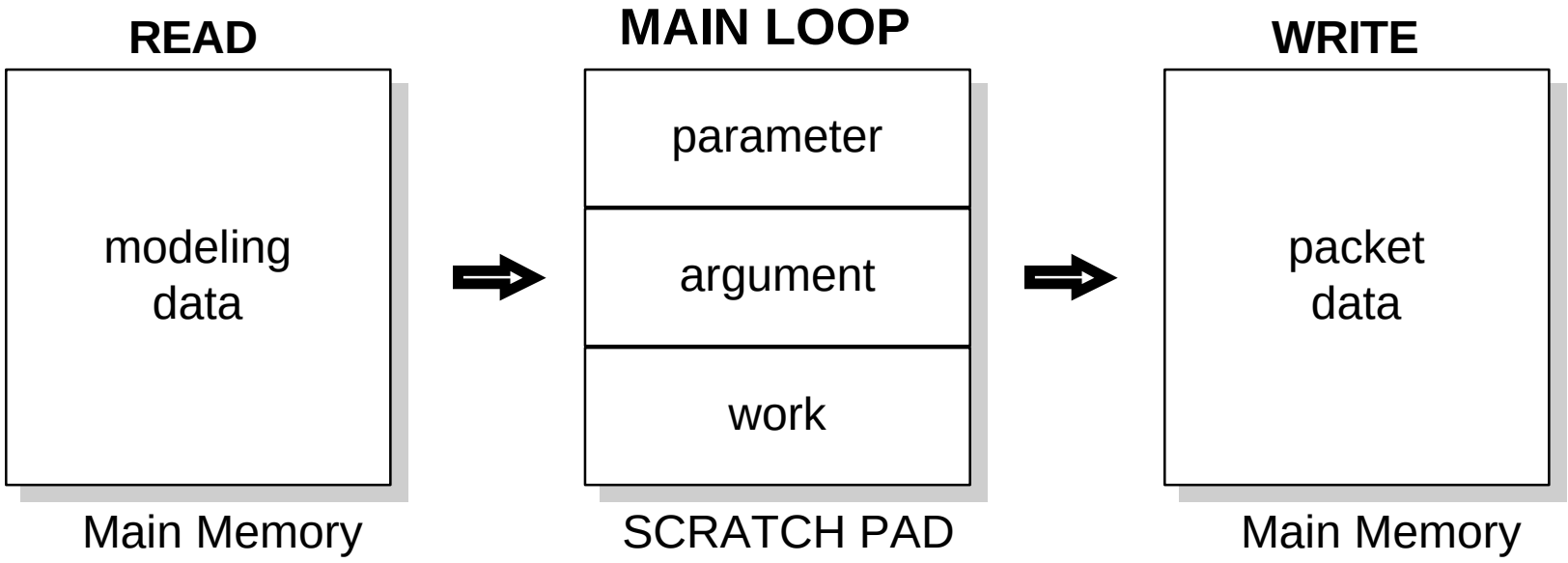


Instruction cache (3)

Use subroutines, keep code short



Data cache (scratch pad) (1)



Data other than modeling data and packet data should be kept in the scratch pad as much as possible

Data cache (scratch pad) (2)

PARAMETER

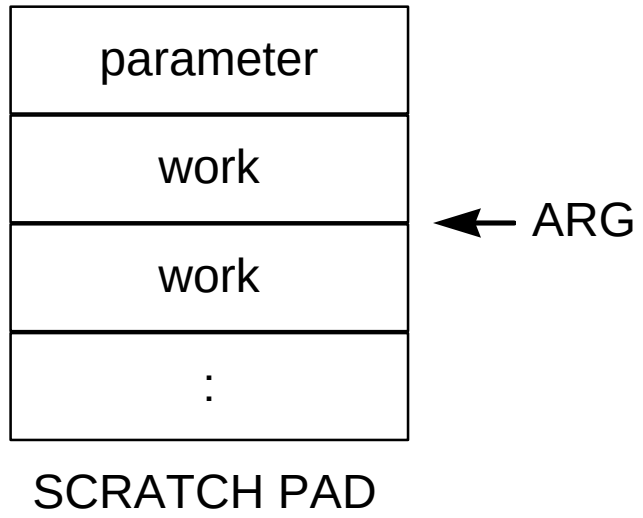
- grouped together and defined as a structure

WORK

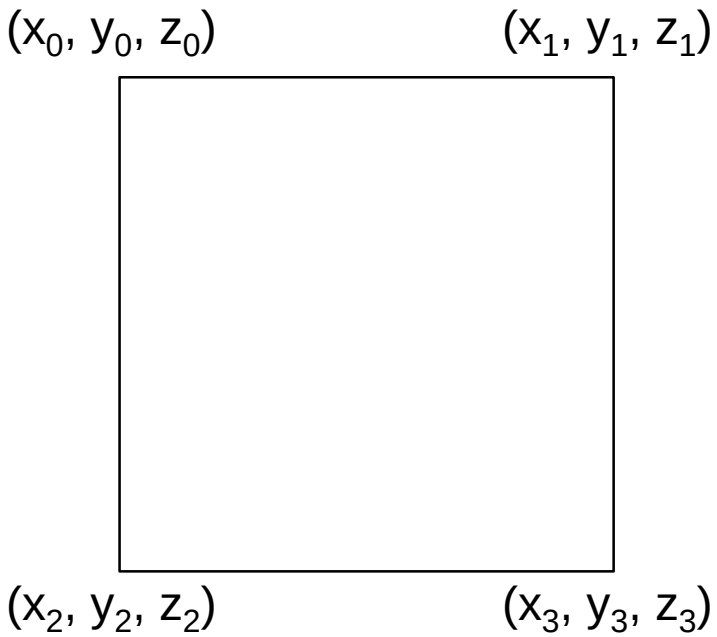
- grouped together and defined as a structure

ARGUMENT

- pointer to the scratch pad



READ modeling data

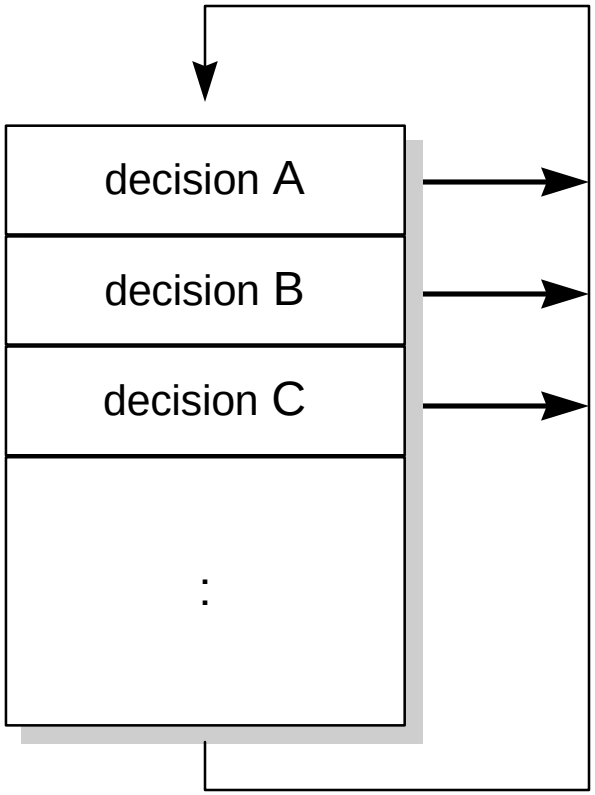


Reading 4 vertices, 12 words takes about 70 cycles

If the data can be expressed as w,h,d, about 7 words and 20 cyclescan be saved

Modeling data formats should take into consideration the fact that memory reads are very slow

Polygons that will not be displayed should be rejected early on



MAIN LOOP

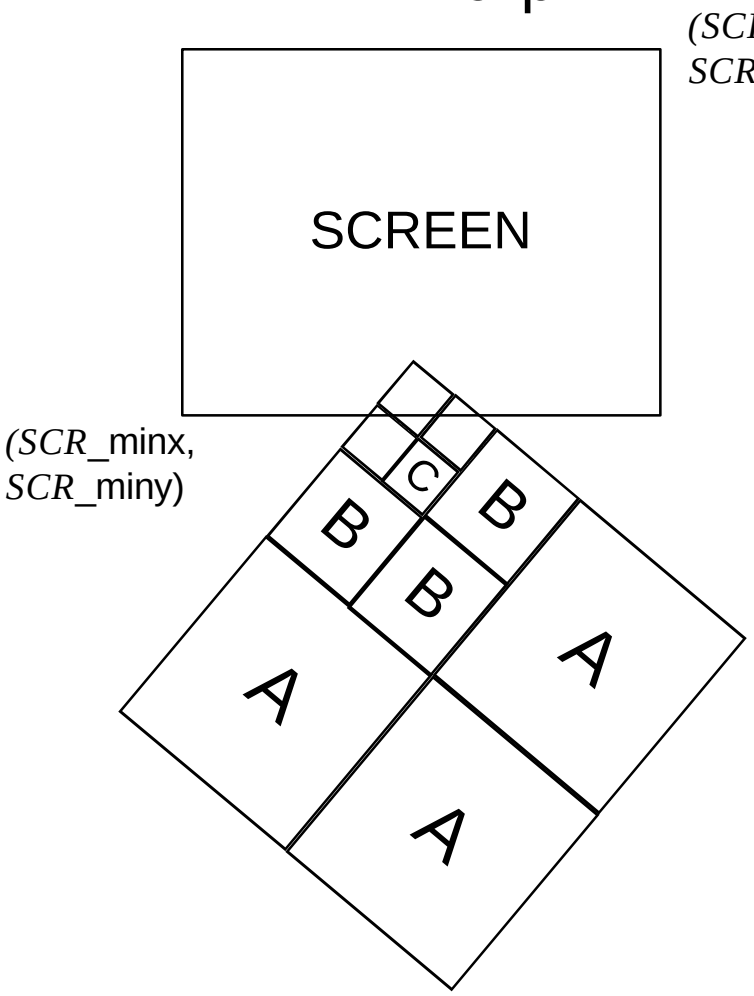
the rejection amount is

$$A > B > C$$

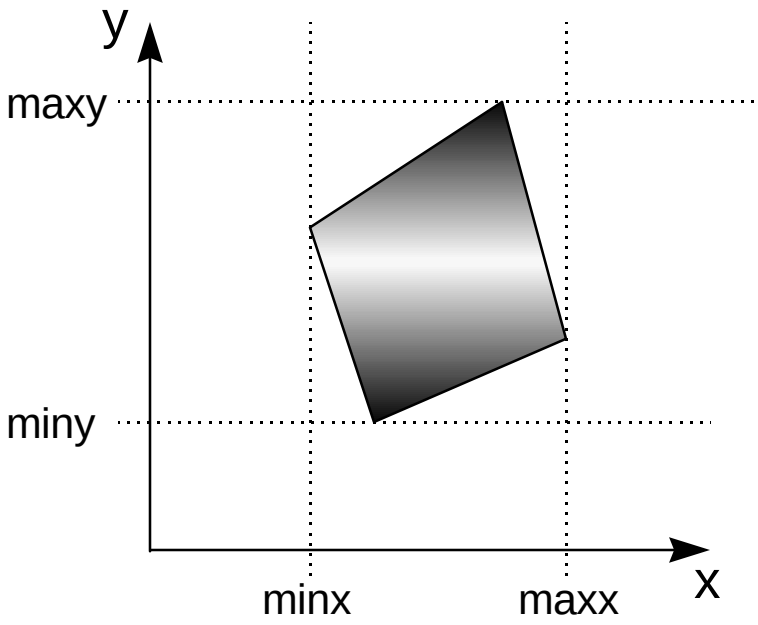
A is the GTE flag clip

Clipping (1)

HW clip



4-vertex **min-max**

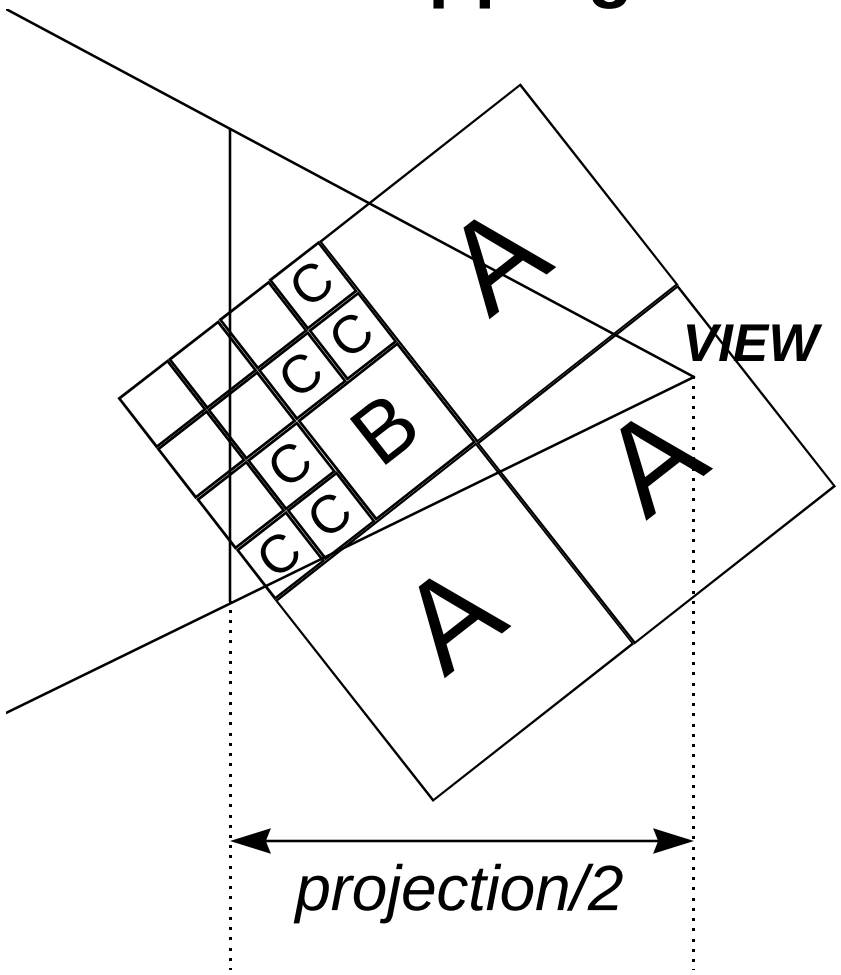


Clip conditions

maxx	>	*SCR*_minx
maxy	>	*SCR*_miny
minx	>	*SCR*_maxx
miny	>	*SCR*_maxy

Clipping (2)

NEAR **Z clipping**



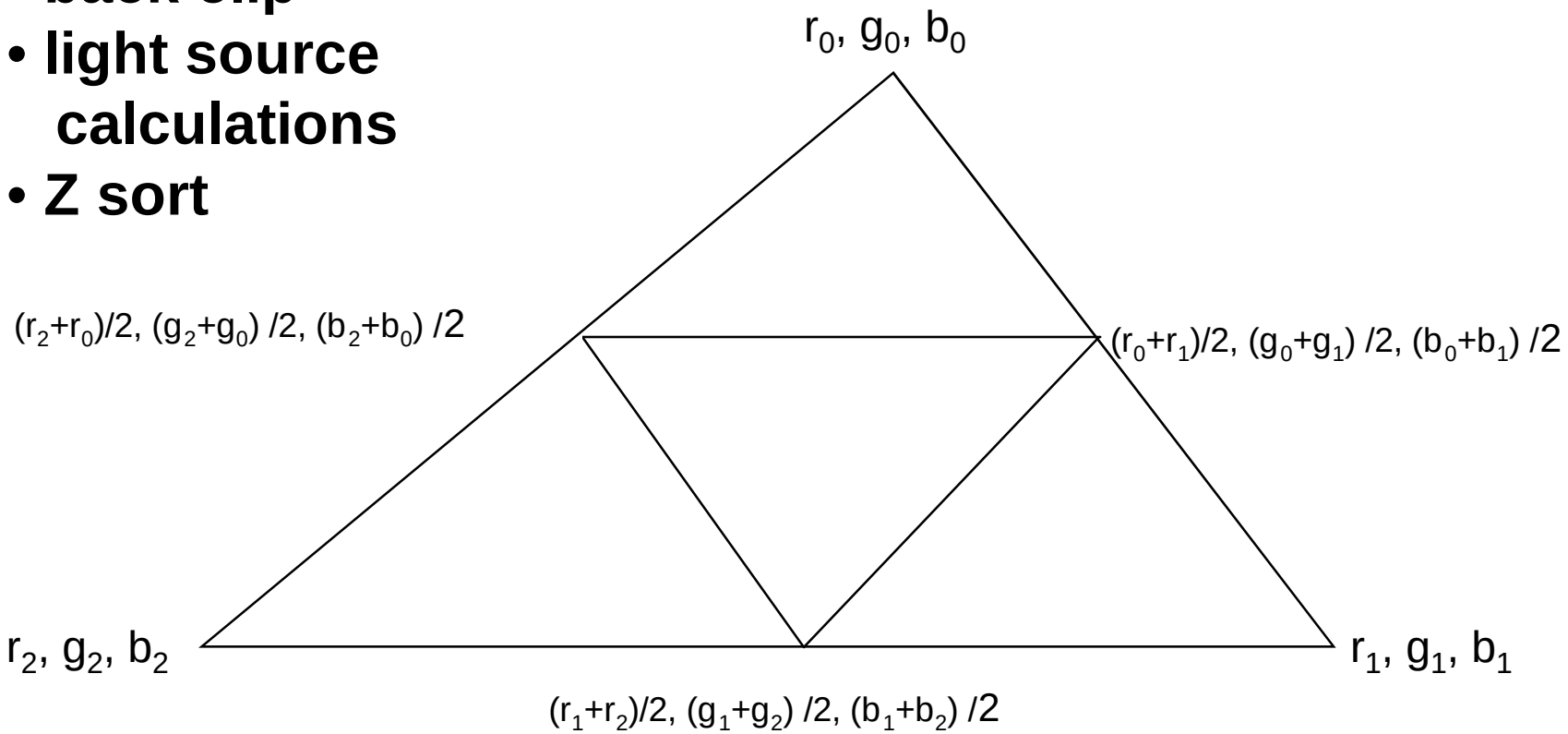
Clip conditions

SZ0 < *projection/2*
&
SZ1 < *projection/2*
&
SZ2 < *projection/2*
&
SZ3 < *projection/2*

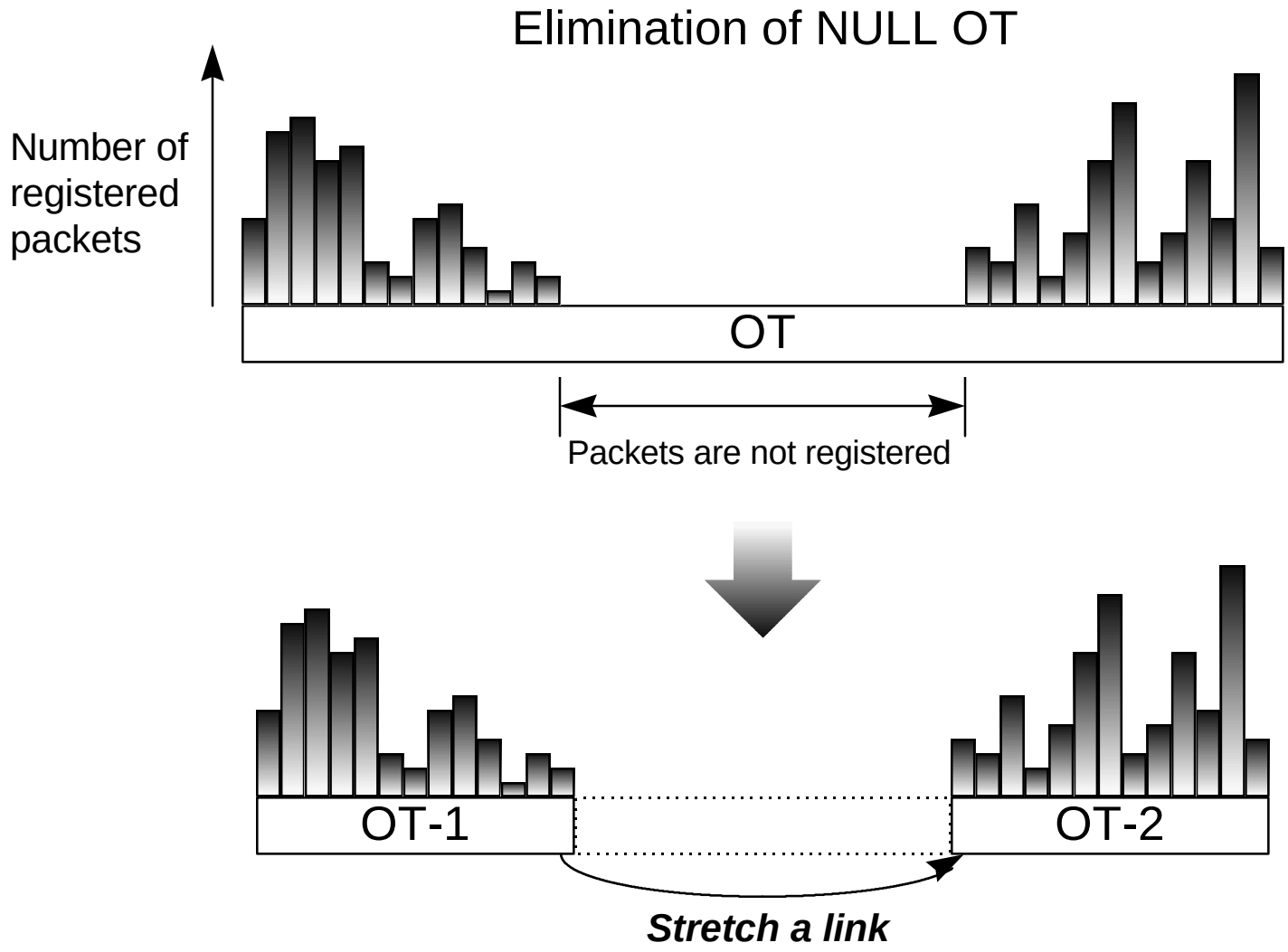
Split processing for before and after division

Processing that is performed just once before division

- **back clip**
- **light source calculations**
- **Z sort**



Eliminating useless OT



Conclusion

Rendering ground in 3-dimensions

1. Active 3-dimension divisions
2. Recursive call
3. On cache