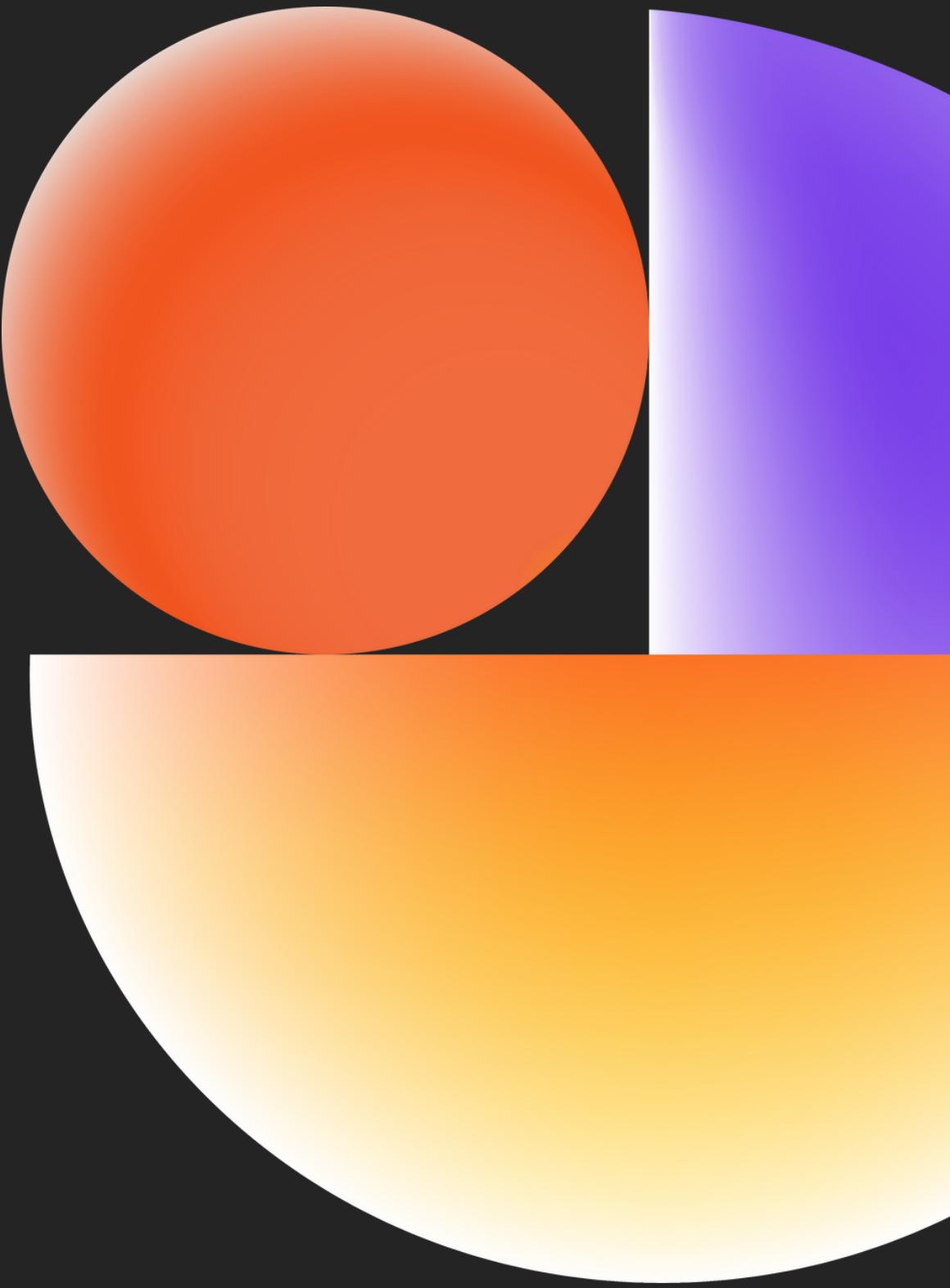


# Sockets

# Laboratório 2

João Pedro Moura Oliveira - 190030879



# A arquitetura usando *sockets*

- █ Utilizando a Camada de Rede
- █ *Portmapper* e o mapeamento de portas
- █ Linguagem de Definição de Interfaces (IDL)
- █ Conversões de tipos e controle da transmissão

# O Problema

$$v[i] = \sqrt{\left(i - \frac{\text{tamanho\_do\_vetor}}{2}\right)^2}$$
$$v[i] = \left(i - \frac{\text{tamanho\_do\_vetor}}{2}\right)$$

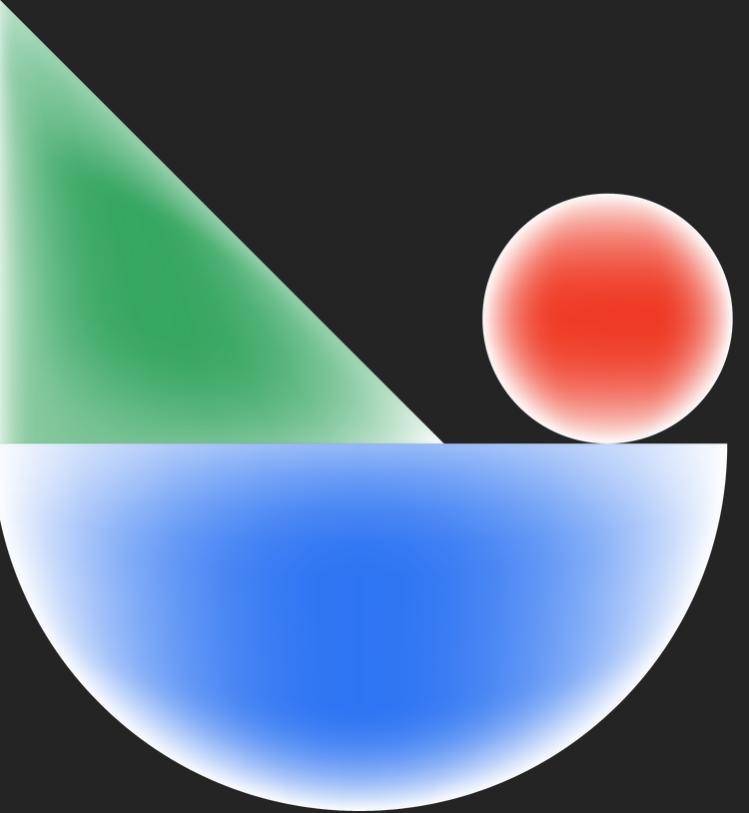
# UDP vs TCP

- Conexão confiável ou rápida ?
- Orientado a conexão ou não ?
- Conexões *Full-duplex*

# 1 Worker e 1 Client

```
void
generate_vetor (vetor *argp, int tam, int inicio, int final)
{
    int tamanho_v = final - inicio;
    argp->vetor_len = tamanho_v;
    argp->vetor_val = malloc (sizeof (float) * tamanho_v);
    printf("[VETOR] => Criando vetor com Inicio = %d, final = %d e tamanho = %d\n", inicio, final, tamanho_v);

    for (int i = 0; i < tamanho_v; i++)
        argp->vetor_val[i] = ((inicio + i) - tam / 2);
}
```



```
// Envia dados para os sockets ja conectados
int tam = atoi (argv[3]);
generate_vetor (&bufout, tam, 0, tam);

int tmp = htonl(bufout.vetor_len);
send (sd, &tmp, sizeof (tmp), 0);

for (int i = 0; i < bufout.vetor_len; i++)
    send (sd, &bufout.vetor_val[i], sizeof (float), 0);
```

# 1 Worker e 1 Client

```
// Recebe informacoes do Cliente
recv (sd, &tmp, sizeof (tmp), 0);
bufin.vetor_len = ntohs(tmp);
printf("[%s:%u] => Recebido tamanho do vetor: %d\n", inet_ntoa (infoCli.sin_addr), ntohs (infoCli.sin_port), bufin.vetor_len);

bufin.vetor_val = malloc (sizeof (float) * bufin.vetor_len);
printf("[%s:%u] => Recebido vetor: ", inet_ntoa (infoCli.sin_addr), ntohs (infoCli.sin_port));
recv_vetor(sd, &bufin);
```

```
void
recv_vetor (int sd, vetor *argp)
{
    recv (sd, &argp->vetor_val[0], sizeof (float), 0);
    printf("%f", argp->vetor_val[0]);
    for (int i = 1; i < argp->vetor_len; i++) {
        recv (sd, &argp->vetor_val[i], sizeof (float), 0);
        printf(", %f", argp->vetor_val[i]);
    }
    printf("\n");
}
```

# Vários *Workers* e 1 *Client*

```
int tmp = htonl(final - inicio);
send (sd, &tmp, sizeof (tmp), 0);

// Envia dados para os sockets ja conectados
vetor bufout;
generate_vetor (&bufout, tam, inicio, final);
send (sd, bufout.vetor_val, sizeof (float) * bufout.vetor_len, 0);
```

```
// Recebe informarcoes do Cliente
recv (sd, &tmp, sizeof (tmp), 0);
bufin.vetor_len = ntohl(tmp);
printf("[%s:%u] => Recebido tamanho do vetor: %d\n", inet_ntoa (infoCli.sin_addr), ntohs (infoCli.sin_port), bufin.vetor_len);

bufin.vetor_val = malloc (sizeof (float) * bufin.vetor_len);
printf("[%s:%u] => Recebido vetor: ", inet_ntoa (infoCli.sin_addr), ntohs (infoCli.sin_port));
recv (sd, bufin.vetor_val, sizeof (float) * bufin.vetor_len, MSG_WAITALL);
```

# Comparativo entre soluções

Quantidade de workers	Tempo gasto (s)	CPU utilizada (%)
2	4,978 total	84
4	4,904 total	86
6	4,847 total	86
8	4,898 total	87
10	4,939 total	86

